The University of Alberta
Printing Department
Edmonton, Alberta

THE UNIVERSITY OF ALBERTA

DISCRETE CHANGE SIMULATION - TWO NEW PROGRAMMING SYSTEMS

by

Robert H. Huculak

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

July, 1968

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled DISCRETE CHANGE SIMULATION - TWO NEW PROGRAMMING SYSTEMS submitted by Robert H. Huculak in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

One of the greatest problems faced by designers of simulation languages is producing a language whereby a user can construct a model which adequately represents the real system being simulated. This thesis will review the problem and illustrate the attempts made to solve it by reference to existing simulation languages.

Two new simulation programs will then be described and illustrated. The first, running in the conventional batch stream, presents a simplified approach to the grouping of model variables. The second represents a new approach to the construction of simulation models as it operates within a time-sharing environment.

ACKNOWLEDGEMENTS

I express my appreciation to Professor H.S. Heaps for his advice, criticism and guidance during the preparation of this thesis; to Dr. G. Syms for his helpful suggestions to improve the final draft; and to Mrs. J. White for time spent in typing the thesis.

I also wish to thank the National Research Council of Canada for providing summer financial assistance to complete this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

One consequence of the development of the modern computer is that simulation has become an important technique for solving problems in many areas. These areas range from the study of the smallest molecules to the most complex interactions of man and space.

Although simulation has proved to be an important and valuable tool for use with a computer, it is not a product of twentieth century technology. For example, Lewis (1967) presents examples of simulation experiments conducted in each of the 17th, 18th, and 19th centuries. However, application of the technique to study the dynamic behaviour of large and complex physical systems did not occur until the development of mathematical techniques such as Monte Carlo Methods (Hammersly and Handscomb (1964)). Although use was made of these methods before the advent of computers (Farmer and Collcut (1967)), the feasibility of their use was greatly increased by the availability of computers.

As indicated by Wallace (1967) simulation, as we shall define it, had its origins in the study of systems, a term which may be loosely defined as a collection of

parts capable of interacting in a manner that satisfies a specific set of requirements. The system may be real or hypothetical, and may be composed of components or subsystems which may be permanent or temporary members of the system. At any given instant the condition of a system is determined by the state of its subsystems. Thus, the state of a system at any two instants is the same if and only if the state of all components of the system is the same at the two instants considered. A state history of a system may be defined in terms of a given time interval and is the set of conditions of the system at each of a set of successive time intervals. A partial state history of the system is obtained by collecting the conditions of the system at selected instants or by choosing the conditions of selected subsystems. A model is a representation of the system under study. It may be a physical model, mathematical model, or symbolic model. The state history obtained for the model is regarded as a state history of the system.

We can then define the art of simulation as the construction of a model of a system and the collection of a state history of the model.

1.2 Early Developments of Simulation Models

One of the earliest developments in the subject of simulation was the separation of simulation models into two distinct groups, continuous change and discrete change models.

1.2.1 Continuous Change Models For many years the differential equations derived in the physical sciences and engineering have been solved by use of analog computers. Thus, systems which may be modelled by suitable differential equations may be simulated by analog computation. However, problems associated with analog equipment include scaling of variables and non-linearity of equations. This has led to the development of digital analog simulators, i.e. programs written for digital computers but which provide the basic operations of analog computers such as summers, integrators, multipliers, etc. Excellent comparisons of various continuous change languages for digital computers may be found in Brennan and Linebarger (1964, 1965), and Clancy and Fineberg (1965).

1.2.2 Discrete Change Models The present investigation will deal with the second type of simulation model, the discrete change model. Teichroew and Lubin (1966) suggest that they may be characterized by the following properties:

1. The system is composed of subsystems of components, each of which has a certain prescribed function to perform.
2. Certain items flow through the system from one subsystem to the next. Each item remains at a specific subsystem until its function is completed; it is then transferred to the next subsystem.
3. Each subsystem has a capacity which cannot be exceeded. Thus, items may have to form a queue before being accepted by a subsystem.

The first programs that utilized computers for simulation studies were written to simulate one specific system in the language currently implemented on the computer. This language was often an assembler language. With the advent of compiler languages, such as FORTRAN, the programming was somewhat simplified. However, since the development of simulation models often involved many changes and consequent reprogramming, it soon became apparent that the special purpose models were not sufficiently flexible.

This led to the development of general purpose simulation languages. At first these languages were very restrictive in their field of applicability. For example,

some were developed particularly for job shop simulations (IBM (1960)), inventory management simulations, and military operations. Then in 1959 one of the first of the true general simulation languages, MONTECODE, was introduced in England (Kelley and Buxton (1962)). Since then many general simulation languages have appeared and, as would be expected, each one has offered some different approach to the problem.

As indicated by Kiviat (1967), with the appearance of general purpose languages simulation theory was soon divided into two schools, the flow chart approach of GPSS (Gordon (1962)) and the statement language approach of SIMSCRIPT (Markowitz, Hausner and Karr (1963)).

The GPSS user specifies the simulation model by constructing a block diagram of the model. Each block is selected from an assortment provided by GPSS, and only these blocks may be used. The user does not specify exactly what happens in each block, but provides certain parameters which act to specialize the action of each block.

The models constructed from SIMSCRIPT, while often based on a flow diagram, are built up using program statements as in other higher level languages. Although the flow chart approach is easier to learn it is not nearly as flexible as the statement approach. For this reason

most simulation languages use the latter approach.

1.3 Purpose of the Study

In the few years since the introduction of general purpose simulation languages, a start has been made to develop a theory of simulation. The present thesis will review the theory which has developed and will illustrate the important points with reference to existing languages.

Two new simulation programs will then be discussed. The first is written in FORTRAN and is designed to run in normal batch mode. The second is written in APL, an automated version of a notation which was first described in Iverson's "A Programming Language" (1962). It is implemented in a time sharing environment and represents a new approach to the development of simulation programs.

CHAPTER II

THEORY OF DISCRETE CHANGE SIMULATION

2.1 Requirements of a Simulation Language

A simulation language can be distinguished from other computer languages by the special features or services it must provide to a user who writes a computer program based on a model of a system. Numerous authors have presented their estimates of what form these distinguishing services should take (Krasnow and Merikallio (1964), Farmer and Collcut (1967), Krasnow (1967), Teichroew and Lubin (1966), Jones (1967a), and Kiviat (1967)). The ones most frequently mentioned are listed below.

1. Since the model is represented in the computer as a set of numbers which describe the state of the model at a particular instant, the language must provide a means whereby this data set can be structured into variables, data, queues, etc., each corresponding to some recognizable entity of the system being simulated. For example, SIMSCRIPT allows a description of the state of a model in terms of entities, attributes of entities, and groups of entities called sets.

Krasnow and Merikallio (1964) refer to these as status descriptors.

2. The simulation language must contain certain instructions with which the programmer can change the state of the model. These include instructions for moving members of a set, arithmetic instructions, and instructions to change values of the data base.
3. The sequence in which state changes are to occur is often highly complex and unpredictable. Thus, each language should contain a built-in timing and scheduling mechanism.
4. Facilities to provide statistical and other performance data, and to produce meaningful output.
5. An important requirement of any computer language is the debugging facilities. These are especially important in simulation programs since use of random number generators and probability distributions make reproducible runs difficult to obtain.
6. Facilities for generating random numbers and for sampling probability distributions. GPSS contains excellent facilities for user and/or system supplied functions.

It must be noted that the above six features cannot be regarded as exhaustive or necessary in every language. However, the manner in which the language implements these features is important and this will be discussed in the following sections.

2.2 World View

The world view of a simulation language is the conceptual foundation upon which the system may be modelled (Krasnow and Merikallio (1964)). It is the way in which the language arbitrarily divides the resources and concepts of the real world to provide the services listed in Section 2.1. To the user it represents the way in which he must think about his problem in order to use the language to develop a simulation model.

In addition to providing the services previously mentioned, the language must resolve certain aspects of the real world and take them into account in its world view. Krasnow (1967) lists the following four characteristics of the real world the simulation language must pay particular attention to.

1. Continuity The real world is made up of continuous actions while the digital computer is capable of producing only discrete changes.

2. Uniqueness It would be impossible to write a program describing every unique action of the real world. It is necessary to isolate basic repeatable patterns and use these as repeatable units in the simulation program.
3. Interdependence It is impossible to account for the mutual interactions of all activities or events in the real world. The language designers must then determine how much interaction of the real world will be modelled.
4. Simultaneity At any instant in the real world there occur infinitely many events. The digital computer, however, can perform one or, at most, a few operations at one time.

The remainder of this section is devoted to a discussion of some of the basic concepts which are used in describing the world view of a simulation language. It is based on papers by Krasnow (1967) and Kiviat (1967) which contain comprehensive discussions of the concepts.

The writing of computer programs as subroutines or procedures has proved to be an efficient and meaningful method. Although actions and events are unique in the real world it has been found essential to provide the same facility in simulation languages. This facility is

used by recognizing certain patterns of behaviour in the real world and differentiating each occurrence by specifying parameters.

To illustrate the use of patterns of behaviour consider automobiles entering a service station for gasoline. The pattern of car entering, possibly waiting in line, getting served, and leaving is repeated for every car.

This pattern can be written as a subroutine or set of subroutines with parameters specifying gasoline type and quantity. These basic modules are referred to as descriptive units of the language, and represent the repeatable patterns of some activity in the system being modelled. An activity refers to the description of the state changes which occur within a descriptive unit.

We are now faced with the problem of resolving the concept of a descriptive unit with the fact of a continuously changing and interacting real world. Two related considerations may be mentioned. First, in the real world no action occurs without the passage of time. However, a computer is capable of producing one state change at a time, and it is accomplished without the passage of simulated time. Second, a state change caused by one activity may influence the action of another activity to perform another state change.

To check for all implications after a state change would add considerably to the programming burden of the user and implementer, since a descriptive unit could contain only one state change. The problem is resolved by making a number of state changes within a descriptive unit, all at the same simulation time, until a plausible interaction point is reached.

If all the state changes within an activity occur at the same simulation time, and terminate with an interaction point, an execution or instance of such an activity is an event. In 1964 with the announcement of SOL (Knuth and McNeley (1964)), the concept of an instance of an activity was extended to include activities in which the state changes occur over a period of simulated time. An instance of such an activity is a process. Then if an activity has only a single interaction point, event and process are identical, and event is a special case of a process. Note that in the above discussion activity refers to the state changes which can occur while event and process refer to the execution of the activity.

2.3 Machine-Based vs. Material-Based World View

The larger differences in existing simulation languages can be traced to the type of descriptive unit used and how they are used to develop a world view. Two separate

approaches can be recognized, the machine-based and material-based approach. Each will be discussed with examples of existing languages illustrating each approach.

2.3.1 Machine-Based Approach In this approach the language allows the programmer to segment the program into activities (usually subroutines) which contain instructions to make state changes. At every simulation time an attempt is made to execute each of the activities in turn. The execution of the state changes in an activity is controlled by a set of test statements located at the beginning of each activity. If all conditions are met the rest of the activity is executed, otherwise the next activity is immediately attempted. After no further activities can be executed at a simulation time, the clock is updated and the procedure is repeated. Note that this type of organization requires no separate scheduling algorithm aside from a control program which branches to each activity in turn.

Updating the clock can be handled in either of two ways; using a fixed time increment or advancing the clock to the time the next event is to occur. The latter approach will usually provide a more realistic simulation (Evans, Wallace and Sutherland (1967)) but will require a scheduling algorithm which scans a list of event notices to find the next event time.

2.3.1.1 Control and Simulation Language CSL (Buxton and Laski (1962)) is an example of a machine-based language which advances simulated time by scheduling events. The language provides the programmer with special status descriptors called T-cells, which are a special type of integer variable. If, for example, the programmer required a variable SHCHANGE to account for a shift change in a simulation, he would define it by specifying TIME SHCHANGE, and in an activity it would be referenced by T.SHCHANGE.

The function of the T-cells in updating the clock can be described as follows. After all activities have been executed (or attempted) at a given simulated time, a scan is performed on all positive valued T-cells to find the minimum. All T-cells are reduced by the minimum value and the clock is advanced by this value. Thus, at least one T-cell will become zero and, during the next execution of the activities, if an activity only contains a test for this T-cell = zero, the activity will be completely executed at this updated clock time.

2.3.2 Material-Based Approach Two methods have been used to implement the material-based approach, the event orientation of SIMSCRIPT and the process orientation of SOL and SIMULA (Dahl and Nygaard (1966)). Although the event was previously described as a special case of the

process, examples of each will be described since the languages resulting from the two approaches are quite different. Regardless of the orientation used, the material-based approach differs from the machine-based in requiring a control algorithm which scans over a schedule of future events to determine the next state changes (note that in the machine-based language CSL a scan was made only to determine the next clock time, not which activity was to be executed next). Since each future event must have a time associated with it, this scan also determines the next clock time.

2.3.2.1 SIMSCRIPT The purest event oriented language is SIMSCRIPT. The programmer constructs a set of descriptive units called EVENT ROUTINES (similar to FORTRAN subroutines), the execution of each constituting an event as previously defined. An event is scheduled by issuing an event notice, and specifying a time at which the event is to occur. This is accomplished by the statement CAUSE (event notice) AT (time). After the execution of an event routine control passes back to the timing or control routine which scans the list of scheduled event notices to find the next scheduled event. The clock is updated, this event notice is deleted from the list, and the event routine associated with this event notice is executed. The procedure continues until no event notices are scheduled.

2.3.2.2 Simulation-Oriented Language The introduction of SOL in 1964 provided simulation programmers with a new approach in the construction of simulation programs. A SOL program consists of a set of descriptive units which in our notation can be classed as activities (the authors refer to them as processes). The activities describe the actions performed by certain entities of the real world, with the programmer having complete control over the number, definition, and detail of the activities.

For example, to simulate traffic flow, a SOL program might consist of one activity describing the movement of automobiles, another describing the movement of trucks and buses, another describing the action of traffic lights, and another describing the movement of pedestrians. All activities are considered to be operating in "parallel" with all others, with interaction being achieved through changing of global variables which are accessible by every activity.

As in GPSS, scheduling and timing in SOL are aided by use of transactions which can be used to represent the flow of a temporary entity through the model. Since the transaction is the occurrence of an activity, it can be called a process. Then, for example, a transaction moving through the automobile activity in the previous example would represent an automobile moving through the model.

There may exist any number of transactions in a many-to-one relationship with an activity, each executing different statements within the activity. Each transaction has associated with it a time variable, a pointer to where the transaction is located in the activity, and local variables.

2.3.2.3 SIMUlation LAnguage The designers of SIMULA recognized the process concept of simulation languages as similar to procedures of algorithmic languages such as ALGOL (Naur (1963)), and consequently extended an ALGOL 60 compiler to allow simulation programs to be written. One of the major changes necessary was to allow "parallel" execution of the simulation activities instead of sequential execution as in the usual implementation.

As with SOL the model is written as a set of activities; the execution of an activity being a process. Once a process has been initiated it may be in one of four states; active, passive, suspended, or terminated. Only one process can be active at one time, corresponding to the movement of only one transaction in SOL. When a process reaches a point in an activity which causes its state to change from active to passive or suspended, the control program automatically issues an event notice for that process, associates a time for next event with it, and places it in the sequence set, a schedule of next events.

It also associates a reactivation point which specifies the location within the activity at which the process is to begin executing when it next becomes active.

2.4 Summary

As is evident from the preceding discussion there exists considerable variety in the approaches used to resolve the dynamic aspects of the real world. Of those developed so far the process approach of SOL and SIMULA appears to be the most flexible, and in a recent paper by Blunden and Krasnow (1967) the authors propose the process concept as a basis for a new simulation language. However, as indicated by Teichroew, et al (1967), 'there is as yet no conclusive evidence that one simulation language is best for a variety of problems'.

There are a number of papers which compare the languages mentioned in this chapter (and others) and the approach used by each in detail. See Tocher (1965), Krasnow (1967), Teichroew and Lubin (1966), and Krasnow and Merikallio (1964), all of which also contain excellent bibliographies. For a detailed description of a particular language, see the reference manuals available for that language.

CHAPTER III

A MACHINE-BASED SIMULATION PROGRAM

3.1 Introduction

This chapter will describe a simulation program which uses the machine-based world view as described in Section 2.3.1. The program is written in System 360 FORTRAN IV source language and a complete listing has been included in Appendix I.

It is supposed that the system to be simulated can be represented by a flow diagram consisting of two basic components, blocks of entities and activities. A segment of a typical flow diagram, showing the two components, is given in Figure 1.

The blocks of entities are used to record the state of variables, required in the simulation model, at various locations in the system being modelled. The position of the variables on the flow diagram corresponds directly to the position of the entities they represent in the real system. For example, in Figure 1, the first entity in each block may represent the number of automobiles present at that particular location in the model, while the second entity may be used to represent the state of traffic lights at the same locations. Changes in the state of the model are recorded as changes in the values

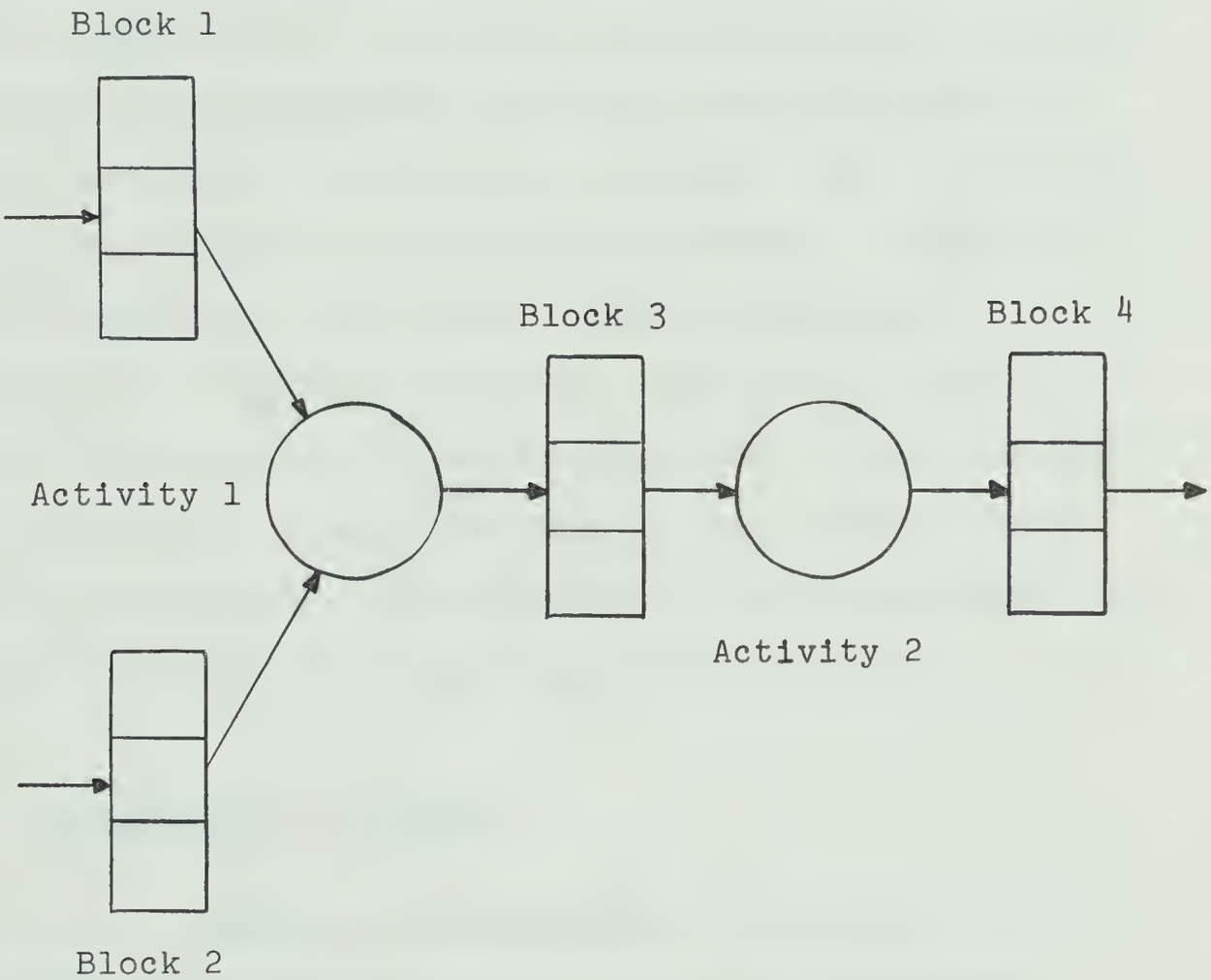

FIGURE 1

of the state entities in some, or all, of the entity blocks. The user must assign the function of each entity required in a model as it is being developed.

As indicated in Figure 1 the blocks of entities are joined to activities. An activity contains, primarily, instructions to change the state of various entities. The programmer indicates which entities are to be changed by linking those entities with an activity which will cause the desired state changes. After the flow diagram has been constructed each activity is written as a FORTRAN subroutine subprogram. Details of their construction will be presented in a later section.

Simulation time is recorded in the variable TIME and is available to all activities. Time is advanced by means of a fixed time step supplied by the user.

3.2 Details of the Program

3.2.1 Entities and Variables As indicated in the previous chapter any system to be simulated contains certain entities and/or variables which describe the state of the system at a particular time. In the present program all variables and entities required to define the system must be in one of the entity blocks. The entity blocks are numbered sequentially beginning at one, and the individual entities within a block are numbered in

the same way. The entities required are arranged as a two-dimensional array referenced by the programmer as STATE(I,J). The first dimension refers to the entity block number, the second to the entity within that block. The number of entity blocks and entities per block are defined by the user, as will be described later.

3.2.2 Activities

3.2.2.1 Configuration Parameters As described in Section 3.1 each activity may have access to only certain specified entity blocks. The user can write the activities in one of two ways to specify the entities and entity blocks available to an activity. In the first method the user specifies explicitly which entity is of interest. For example, if an activity required the fourth entity in entity block two, the user would reference it in the activity as STATE(2,4).

In the second method the user specifies the entity references as integer variables. The entity references are then known as configuration parameters which are specified at the time the activities are being assembled. The activities can then be written so that they can be used at different places within the model, with each application differentiated by the configuration parameters.

3.2.2.2 Activity Parameters The user has one additional way to specify the unique action of an activity. This is by use of activity parameters. Whereas the configuration parameters are used to indicate the relationship between entity blocks and activities, the activity parameters are used to differentiate between the action of the same activities. For example, an activity may be written which will move a multiple of an entity from one block to another. This subroutine may be used a number of times within the flow diagram. The difference in each case is the value of the multiplication factor. Both the configuration and activity parameters are supplied during the assembling of the model and remain constant for that model.

3.2.2.3 Format of Activities In constructing the activities it is necessary to impose a number of restrictions over and above those which result from using the FORTRAN language. They pertain primarily to the use and definition of some variables. Figure 2 contains an example of a typical activity. The following outlines those features which must be present in each activity to be run using the present program.

1. The subroutine name must be of the form OPNX, where, in the present program, X is an integer from one to sixteen. This allows for up to

```
      SUBROUTINE CPN1
      COMMON/BLKA/STATE,PPAR,KPAR,NAMOP,NKPAR,NPPAR,KIND,NUMB
      COMMON/BLKB/NBLOK,NSTAT,NKP,NPP,KKP,KPP,INDEX
      COMMON/BLKC/NCON,NOPS,MAG,NUSE,NCN,NOUT,NDAT,TIME
      DIMENSION NAMOP(16),KIND(144),NUMB(144),KPAR(100),
     1NKPAR(16),NPPAR(16),STATE(25,40),PPAR(100)
      NKP=3
      NPP=0
      GO TO (1,2),INDEX
1     RETURN
2     I = KPAR(KKP)
      J = KPAR(KKP + 1)
      K = KPAR(KKP + 2)
      IF(STATE(I,J).EQ.0.0) RETURN
      STATE(I,K) = STATE(I,K) + 5.0
      RETURN
      END
```

FIGURE 2

sixteen activities to be defined and used in a model. The name of each new subroutine used must be entered by a call to a special section of the control program, to be described later.

2. The COMMON and DIMENSION statements are required to allow the subroutine and control program access to the data base.
3. The next four statements must appear as indicated in the example, immediately following the DIMENSION statement. The two assignment statements define the number of configuration parameters (NKP) and activity parameters (NPP), respectively, for this activity. The purpose of this set of statements will be described in a later section.
4. The remainder of the activity may be considered to be divided into two sections. The first contains test statements which must be satisfied before the state changes contained in the second section are performed. All the state changes indicated in the latter section must be capable of being executed at the same simulated time.

3.3 Control Program and Command Words

The program offers the user a number of functions which can be performed by use of command words. The

simulation program is run by specifying one of the command words which causes a branch to a certain section of the control program. The program performs the function called for after which it returns and requests another command word.

The control program also handles all input and output, unless it is data requested by an activity during a simulation run. The program produces output resembling that obtained at a remote terminal of a time-shared system. All input statements are preceded by a line of output requesting the desired information. Immediately after the input is read from cards it is printed out, thus producing a well-defined record of all input to the program.

The program can also be instructed to store a model currently in memory on magnetic tape. This stored model may then be recalled and its operation continued at some later date.

The function and operation of each command word is described in the remainder of this section.

3.3.1 NEWTAPE This command is used to prepare a new magnetic tape by rewinding it and placing on it a single record of nineteen words consisting of -1, 0, 0, 16 blank words. The first zero indicates that no activities have been defined, the second indicates that no models have been stored on the tape, and the sixteen blank words contain

space for the names of up to sixteen activities. This record forms a header record on the magnetic tape. As other records are added to the tape the header record is moved along so that it always remains the last record on the tape and provides a summary of the previous records.

3.3.2 NEWACT This command is used to define new activities. Following input of the command word the program requests the names of the new operations, each containing four characters followed by a blank. The program then assigns each operation a unique operation number chosen sequentially from the first sixteen integers. The header record on the tape is updated, containing the names of all the defined activities.

3.3.3 NEWMOD This command is used to set up a new simulation model. In response to the command the program requests two parameters. The first, NSTAT, is the number of state entities required in each block of entities. The second, NBLOK, is the number of blocks of state entities required in the model.

The user then lists the activities to be used in the model in the sequence in which they will be attempted by the scheduling program. To differentiate between multiple use of the same activity, each activity name must be followed by an integer. The program then establishes in

memory a vector KIND to indicate the order in which the defined activities are to be executed. The components of KIND are the activity numbers that were defined by the NEWACT command. Thus, the activities are identified by name for the user but by number for the computer.

For example, assume the activities LINT, DIFF, and HYDV were entered as the first three activities by the NEWACT command. Then if the sequence of activities is input as

DIFF1 HYDV1 DIFF2 LINT1 DIFF3 ,

the vector KIND will consist of 2, 3, 2, 1, 2.

After the activities and their sequence has been established, the program establishes two additional control vectors NKPAR(I), NPPAR(I), I=1,NTOT, where NTOT is the number of components in the vector KIND and represents the number of activities used in the model. The I^{th} component of NKPAR is the number of configuration parameters required by the I^{th} activity, and of NPPAR the number of activity parameters required by the I^{th} activity.

To evaluate the parameters required in the above two vectors the program branches to each activity in turn as specified in the vector KIND. As shown in Figure 2 the first executable statements in each subroutine assign values to the common variables NKP and NPP. These represent,

respectively, the number of configuration parameters and activity parameters required in this activity. During this phase of using the program INDEX has been set to unity. Thus, the subroutine will be exited immediately and upon return to the control program the I^{th} component of NKPAR and NPPAR will be set to NKP and NPP, respectively. This procedure is repeated until all components have been evaluated.

After the above two vectors have been evaluated the program requests, as input on cards, NKPAR(I) configuration parameters for each activity I and NPPAR(I) activity parameters for each activity I. The configuration parameters are stored in the vector KPAR and the activity parameters in the vector PPAR. At this point the model is completely specified and if all activities have been compiled the model may be run.

3.3.4 STORE This command causes a check to be made as to whether a model is in memory available to be stored. If available the program will create a new record on tape immediately following the last stored model consisting of all information required to retrieve and begin running the model at a later time. The header record is updated indicating an additional model has been stored and placed at the end of the new model.

3.3.5 CALL This command causes a previously stored model to be retrieved from magnetic tape and placed in memory as the active model. Each stored model is referred to by a sequential integer, the first model referred to as one. Then if the user specifies retrieval of model J, the J^{th} model stored on the tape will be obtained, unless J is greater than the number of stored models. If the latter occurs an error message will be output.

3.3.6 RUN This command allows a model, which has been assembled and for which all subroutines are compiled, to be run. The program requests the time increment and length of simulation run as input data. It then branches to each subroutine in the sequence indicated in the vector KIND. After each cycle through KIND the simulation clock is advanced by the time increment supplied by the user. The procedure is repeated until time has been advanced the specified amount.

After completing each cycle through the activities, and before the clock is updated, the control program branches to another subroutine, OUTPUT, which the user is also required to supply. It is through this subroutine that the user obtains the required state history of the model currently being run. Since it is referenced at each simulated time it is possible to obtain a complete state history of the model. This would be quite valuable

in debugging a model. However, it can be used quite selectively as shown by the example in the next section, where the value of a few entities are obtained at the end of a run. A copy of the output subroutine used in the example can be found in Appendix I.

3.3.7 INIT When the control program encounters this command word it immediately branches to a subroutine with the same name, where the user is required to supply a yes or no answer to two questions in turn. The first requires a yes to initialize all state entities to zero and the second a yes to initialize only some state entities, as specified by the user. For the latter the user is required to supply a card for each entity to be assigned a non-zero value. Each card contains the block number, entity number within the block, and the value of the entity, with a zero block number terminating the cards.

3.3.8 END This command signifies the end of data for the simulation program, terminating its execution.

3.4 An Example - Simulation of a Simple Three Machine Assembly System

3.4.1 Description of Problem Figure 3 presents a flow diagram of a system that will be used to illustrate the features of the simulation program presented in this

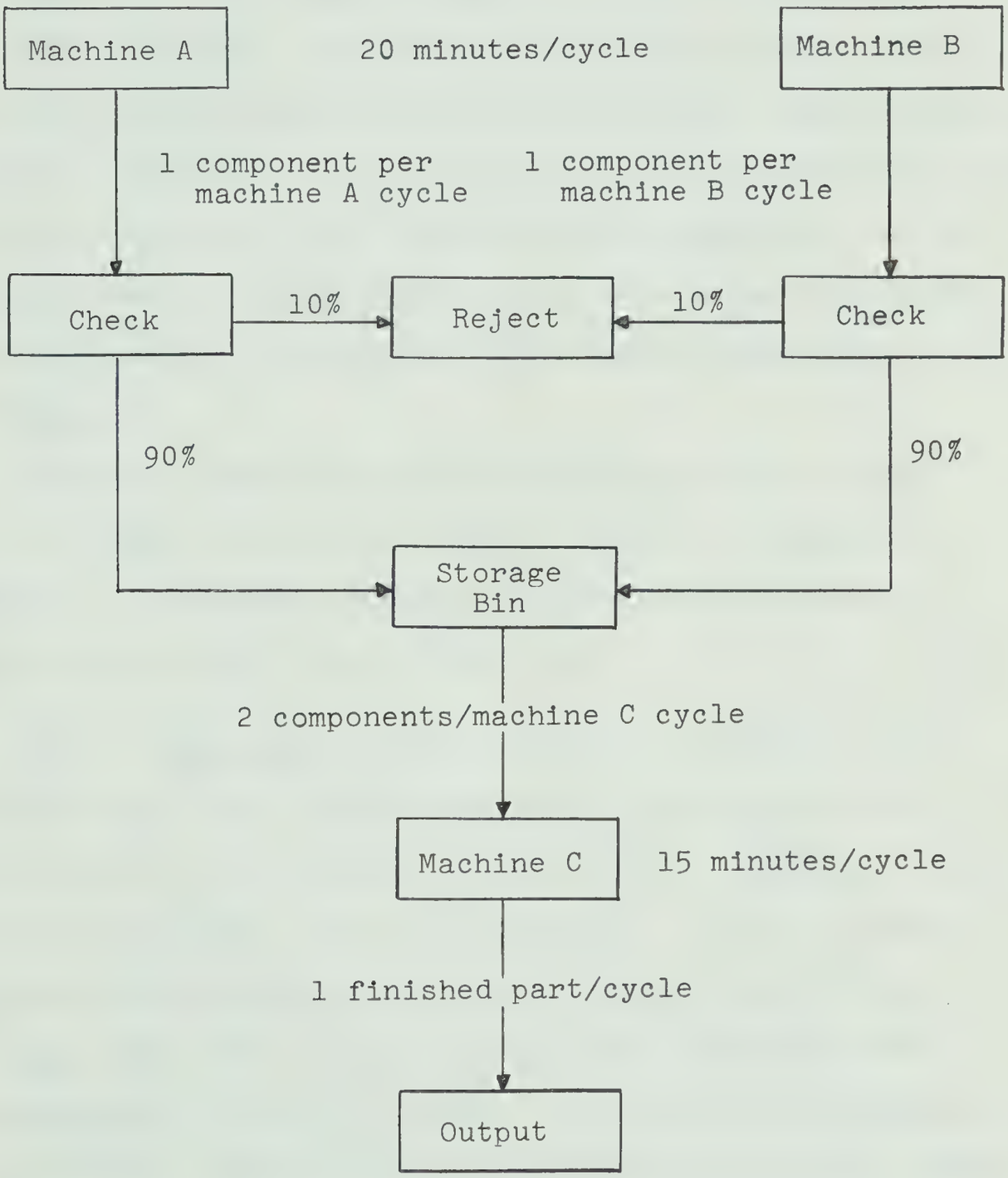
Machine A
20 minutes/cycle
Machine B
1 component per machine A cycle
1 component per machine B cycle
Check
10%
Reject
10%
Check
90%
90%
Storage Bin
2 components/machine C cycle
Machine C
15 minutes/cycle
1 finished part/cycle
Output

FIGURE 3

chapter. It can be regarded as a sub-system of a larger production system which depends on the output of final product from machine C.

Basically the system consists of three facilities, machines A, B and C. Machines A and B are started up at the same time and each produces one component every twenty minutes, for a total of two components every twenty minutes. There is, however, a 10% chance that any component may be rejected due to imperfections. Any rejects are discarded from the process. Components not rejected are placed in a storage bin.

Machine C requires two components from the storage bin to produce a finished product, taking 15 minutes to complete. If the storage bin contains fewer than two components machine C will remain idle.

3.4.2 The Model Figure 4 presents the final entity block-activity flow diagram developed for this problem, and consists of four entity blocks, with a maximum of three entities per block, and five activities. Only four subroutines are required since activities one and five use the same subroutine denoted in the flow diagram as SETF and compiled as subroutine OPN1. It requires the specification of three configuration parameters to completely define it. The other three subroutines do not require any configuration or activity parameters supplied prior to running

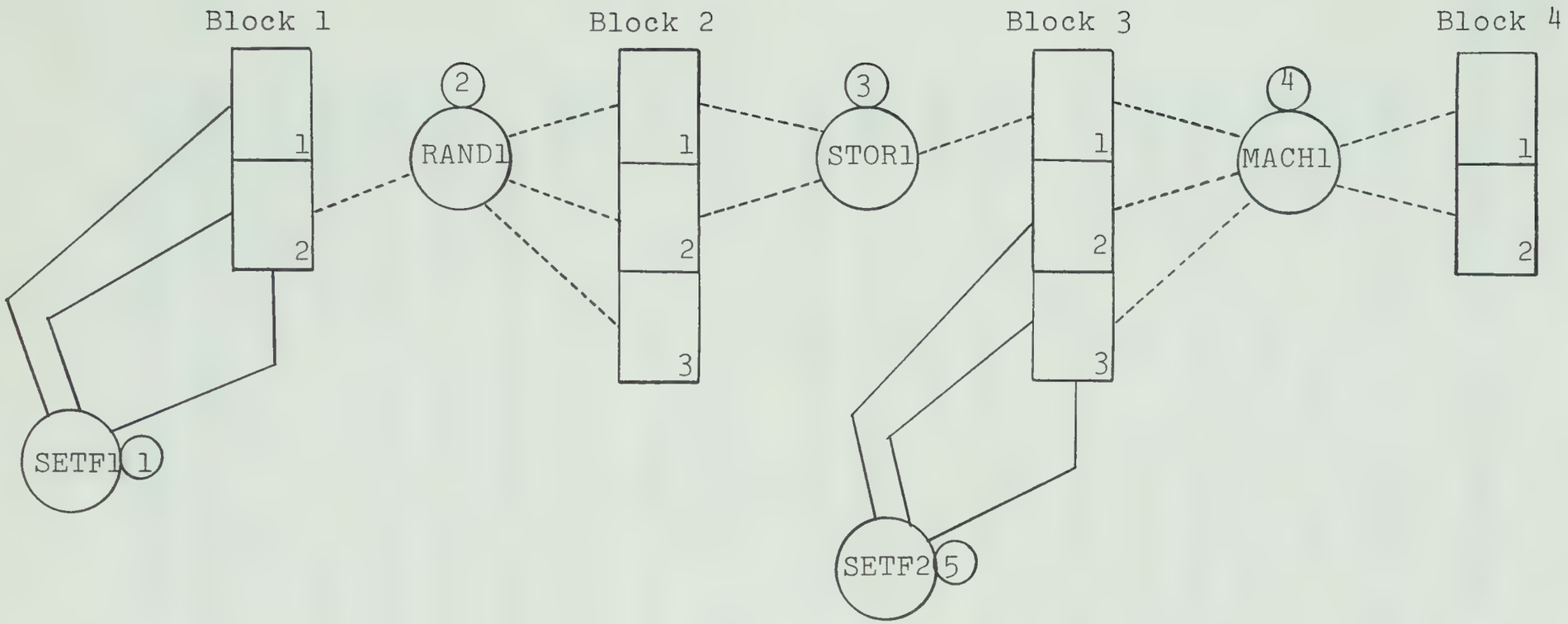
Block 1
Block 2
Block 3
Block 4
1
2
3
RAND1
STOR1
MACH1
SETF1
SETF2
specified by configuration parameters
specified explicitly in activity

FIGURE 4

the model. A listing of the subroutines may be found in Appendix I.

The entities used in the model are defined as follows:

STATE(1,1)	20 minute timer flag
STATE(1,2)	20 minute timer
STATE(2,1)	component from machine A
STATE(2,2)	component from machine B
STATE(2,3)	total components rejected
STATE(3,1)	total components currently in storage bin
STATE(3,2)	15 minute timer flag
STATE(3,3)	15 minute timer
STATE(4,1)	total finished products from machine C
STATE(4,2)	total time machine C in operation.

The following is a description of each activity in the model and of the state changes caused by each.

1. SETF1 The three configuration parameters I, J, K requested by subroutine SETF are 1, 1, 2 respectively in SETF1. This activity simply increases the contents of STATE(1,2) by five whenever STATE(1,1) = 1. Since the flag is never changed this will occur on each cycle.
2. RAND1 This activity acts in conjunction with SETF1 to provide the model with a 20 minute

timer. Whenever STATE(1,2) < 20 no action is taken, otherwise the following occurs. STATE(1,2) is reset to zero and two random numbers, say RN1 and RN2, are generated. If RNi ≤ .9 set STATE(2,i) = 1, otherwise STATE(2,i) = 0, i=1,2. The action of this activity represents the production of two components every 20 minutes. The random numbers are used to determine if a component is a reject or not, signified by placing a zero or one, respectively, in STATE(2,i), i=1,2. For every rejected component increment STATE(2,3) by one.

3. STOR1 This activity adds the completed and acceptable components, in STATE(2,i), i=1,2, to those already in the storage bin, represented as STATE(3,1). STATE(2,i), i=1,2 are immediately reset to zero. Note that this activity can be executed every cycle since nothing will be added to the storage bin unless components are produced by the previous activity RAND1.

4. MACH1 This activity represents the state changes associated with machine C and can be divided into two sections.

   (a) A check is made to determine if machine C is busy. If STATE(3,2) ≠ 1 machine C is

not busy, and go to part (b). Otherwise, check STATE(3,3), the 15 minute timer, to determine if the 15 minute processing time has elapsed. If STATE(3,3) < 15 no further action occurs. Otherwise set STATE(3,2) = STATE(3,3) = 0 and increment the count of completed products by one.

(b) Check if the storage bin contains at least two components to utilize machine C. If STATE(3,1) < 2 machine C cannot be used. Otherwise set machine C to being used by STATE(3,2) = 1 and decrease the amount of components in the storage bin by two.

5. SETF2 The three required configuration parameters are 3, 2, 3. The activity will then increase STATE(3,3) by five if STATE(3,2) = 1. Thus the fifteen minute timer is used only if machine C is busy, and indicates the amount of processing time used on the current set of components.

3.4.3 Results Appendix I contains a listing of the output obtained when the example of this section was simulated. Output originating in the control program is preceded by C) while lines of output corresponding to user supplied data are preceded by U). Any output not prefixed

in this manner originates in the subroutines supplied by the user.

Since no activities or models have previously been defined the user specifies NEWTAPE to initialize the counters of these entities. The next command word specifies that new activities are to be defined. The program then assigns each a subroutine name by which the activities must be compiled. Then, with the activities compiled a new model is set up, containing four entity blocks with three entities per block. The next line of user data contains the names of the activities used in the model in the sequence they are to be executed. Each name is followed by an integer indicating the number of times the subroutine has been used up to this point. Next the configuration and activity parameters are input and, after initializing the entities, the model is ready to be run.

After supplying the time increment and length of simulation run, the model is executed ending with a listing of the output of machine C and the total time machine C was running. As indicated in the listing of the results in Appendix I, in an eight-hour run machine C produced 19 finished products and was in use 290 out of 480 minutes.

CHAPTER IV

OSS - AN ON-LINE SIMULATION SYSTEM

4.1 Introduction

4.1.1 Reasons for an On-Line System As indicated by Dimsdale and Markowitz (1964) and Jones (1967a) the normal course followed in simulation is: (1) writing of programs to model the system, (2) testing of programs, and (3) generation of output from simulation runs. The first two steps, however, are usually repeated many times in order to obtain an accurate model of the system.

Conventional methods of writing simulation programs, using languages such as described in the previous chapter or GPSS, SIMSCRIPT, SIMULA, etc., require the programmer to submit his programs to be run in the off-line batch processing stream. This method usually implies a large number of runs to obtain a valid and debugged model, resulting in the loss of considerable time waiting for the program to be returned from the batch stream.

A recent development in modern computing technology is the technique of time-sharing. Special purpose hardware and software systems allow a number of users at remote terminals to simultaneously have access to a large computer.

On-line computing facilities provide means of

reducing the time required to build and test a simulation model since the building and testing may be done more or less simultaneously. One on-line system for simulation, OPS-3 (Greenberger, et al (1965)), has already been implemented and a new language and system for on-line use has recently been specified (Jones (1967a, 1967b)).

This chapter will describe a new on-line simulation system written in APL and used on the IBM 360/67 at the University of Alberta.

4.1.2 Features of an On-Line System As discussed in Section 2.1 all simulation languages must provide certain features if accurate models are to be produced. In an on-line environment a number of additional features may be considered desirable. Some are listed below.

1. The system should allow the programmer to build and test sections of the model before all the pieces have been put together.
2. An interpretive and compiled mode of execution should be available; interpretive for building and testing the model to avoid recompilations and compiled for running completed programs.
3. Comprehensive on-line debugging and tracing facilities should be included.
4. There should be facilities to save a model at

any state of construction or running and recall it at any later time.

4.2 Description of OSS

This section will present a detailed description of the OSS system. See Appendix II for a complete listing of the APL programs.

4.2.1 General Organization and World View The world view of OSS encompasses the material-based or flow oriented model. It is assumed in this approach that the dynamic behaviour of a system is caused by something moving through the system which acts in some manner to cause changes in the state of the system.

The state of a system modelled in OSS is indicated by two types of entities, permanent and temporary. Permanent entities are present during the entire simulation whereas temporary entities are created and destroyed during a simulation run. In OSS the latter are called transactions, and are used to represent those items which can be thought of as flowing through the system.

A simulation program in OSS consists of a set of activities or functions in APL notation (throughout this discussion the terms will be used interchangeably). Each activity contains instructions for state changes and, since the execution of an APL function cannot be suspended,

to continue at a later time, the state changes in an activity must all occur at the same simulated time. Thus, each execution of a function constitutes an event as in SIMSCRIPT.

The transactions flow through the model by causing the execution of the activities at particular points in time. OSS automatically maintains lists of when the events are to occur, and a special variable, referred to as clock or simulation time, records the instant of real time that has been reached in the simulated model.

All clock times in OSS are integers, the unit of real time corresponding to a unit of simulated time being defined by the user and used throughout a given model.

4.2.2 Transactions Although the use of transactions is most apparent when simulating a system in which physical objects are flowing, in a more abstract view they may be interpreted only as vehicles for scheduling activities (Krasnow (1967)). Either view may be used when planning a model.

Each transaction in OSS consists of a set of integers divided into two classes; a group of six system variables used by the scheduling programs, and a variable number of user defined parameters.

The system variables consist of a transaction priority number, a transaction number, next event indicators, a

time of next event indicator, and a transaction status indicator. The only one directly accessible to the user is the transaction priority number, with facilities provided for the user to assign and change the priority of a current transaction.

During the setting up of a model, to be described later, the user is required to supply the number of transaction parameters required. Each transaction generated in the model will then have the specified number of parameters associated with it. These parameters are generally used to indicate uniqueness among the various transactions which may be in the system at any given time. For example, the transactions may represent ships in a berthing and unloading system, and the parameters could specify variables such as ship size, type of cargo, and time to unload cargo.

The transactions are maintained in two lists by the scheduling program. The current events list contains transactions whose next event times are less than or equal to simulated time. The future events list contains transactions with next event times greater than simulated time.

The current events list contains, primarily, transactions which are in a blocked state, i.e. they cannot execute the next activity due to the unavailability of some required entity. They will, however, be executed as

soon as possible. Also included in this list are transactions which will be moved at the given simulated time, including the current transaction.

Transactions may enter the simulation in one of two ways; either generated automatically by the scheduling program or supplied by the user. To generate transactions automatically, the user must supply the value of two OSS variables, *MEAN* and *MOD*, with *MOD* ≤ *MEAN*. The transactions will then be generated every *MEAN* ± *MOD* time units, with equal probability given to each number in the range. Note that if *MOD* equals zero, transactions will be generated at a constant rate, every *MEAN* time units. All transactions generated automatically are assigned a priority of one, the highest allowed in the system.

The second method permits the user to place any number of transactions on the future events list before a simulation run. However, the user is required to supply all the system and user parameters for each transaction. This second method is useful in testing and debugging a model. The program also allows the two methods to be used simultaneously.

After a transaction has been created it may remain in the system for an extended period of time during which its state may be active, blocked, suspended or terminated. Only one transaction, the current transaction, may be

active at any one time, and the parameters associated with a given transaction can only be referenced at this time. A later section will describe the relationship of the transaction states to the movement of transactions. Transactions which move into the final state, terminated, are not required in the system any longer, and are deleted from the system.

4.2.3 Permanent Entities OSS provides the user with two permanent entities representing two pieces of physical equipment of the real world, facilities and storages. A facility is a piece of equipment which can be engaged by only one transaction at a time. A storage is a piece of equipment which can be engaged by any number of transactions at one time up to some predefined limit. Although a facility can be occupied by only one unit per transaction, a transaction may enter any number of units into a storage, subject to the above condition.

If at any time transactions can not continue through the model, for example, if a facility is occupied by another transaction, they will be placed in a first-in first-out queue advancing one at a time as the entity becomes available. Queue statistics may be obtained in such an instance by using the system supplied functions *QUEUEIN* and *QUEUEOUT*. These provide statistics such as maximum transactions in the queue, total transactions in

the queue, number of transactions not delayed in the queue, and average time in queue.

In addition statistics are automatically gathered for each facility and storage defined in the model when the functions *FACILITYIN*, *FACILITYOUT*, *STORAGEIN*, and *STORAGEOUT* are used. Facility statistics include total number of transactions that used the facility, average time transaction in facility, and fraction of use of facility. Storage statistics include average, maximum and current contents of the storage.

Facilities, storages and queues are identified in OSS by assigning to each entity type successive integers starting at one.

In addition to the above entities the system allows the user to define any additional variables or data sets which may be required in a model.

4.2.4 OSS Scheduling Program One consequence of using a material or flow based world view is the necessity of supplying, as part of the simulation system, a rather elaborate algorithm to schedule the occurrence of events. The complexity varies with how much of the scheduling is left to the user. For example, SIMSCRIPT contains a fairly simple algorithm as the user is required to program the occurrence of each event by means of event notices. However, languages such as GPSS, SOL and SIMULA remove the

burden of the user scheduling all events by use of more elaborate programs. OSS uses the latter approach.

4.2.4.1 Organization An important aspect of the OSS scheduling algorithm is a set of user supplied configuration parameters. These indicate the activity which is to be executed immediately after a given activity, and are supplied as the model is being assembled. In general only one configuration parameter is required for an activity, however, OSS will permit the user to specify a primary and secondary parameter providing the activity contains a *TRANSFER* statement. This statement allows the user to change the path transactions may take through the system and will be described later.

Operating in conjunction with the scheduling program are a number of control statements. They are included in the activities by the user to indicate to the scheduling program the status of the current transaction after an activity has been completed. They include the following statements.

1. *ADVANCE A* The current transaction is to become suspended and placed on the future events list for *A* time units, after which the next activity is to be attempted.
2. *BLOCK* The current transaction could not execute

the next activity. It is kept in the current events list in a blocked state until the blocking condition is removed.

3. *HOLD A* The current transaction could not execute the next activity because of a blocking condition. It is placed on the future events list for *A* time units after which the activity is again attempted.
4. *ENDTRANS* The current transaction is deleted from the current events list, the count of completed transactions is incremented by one, and the simulation limits checked for an end of simulation run.

If none of the above statements are executed explicitly, or by other supplied functions to be described later, the scheduling program maintains the current transaction in the active state and immediately begins execution of the next activity as defined by the configuration parameters.

4.2.4.2 Operation As previously indicated in Section 4.2.2 each transaction has, as one of the system variables, a transaction status indicator. For transactions in the current events list the status indicator has the value one or four indicating the transaction was blocked the last time it was active or the transaction is

available to attempt the next activity, respectively. Note that if all status indicators in the current events list are equal to unity no further state changes can occur at this simulated time.

Assuming the simulated clock has just been updated the scheduling program will proceed as follows. Each transaction in the current events list, beginning with the first one in the highest priority class, will be selected in turn as the current transaction. Each transaction is moved through as many activities as possible until a point is reached at which it can cause no further state changes at the current simulated time. After no further transactions can become current at the simulated time, the scheduling program moves the transaction with the lowest event time from the future to the current events list, where it is inserted as the last transaction within the correct priority class. The clock is updated to the time of this transaction and the process is repeated beginning with the first transaction with the highest priority.

In order to provide a more realistic dynamic model the above sequence is altered under certain conditions. For example, a transaction near the end of the current events list may release a facility which is blocking a transaction near the beginning of the current events list. The moving of transactions would be started over as soon

as the former transaction became suspended. Thus, events will occur at the earliest possible time and in the correct sequence.

4.2.5 OSS Control Program The use of OSS is under control of a main program through which the user specifies the section of OSS he wants to use. The control program will type *ENTER COMMAND WORD* and the user will respond by typing one of *NEWMODEL*, *STORE*, *LOAD*, *RUN*, *NEWRUN*, *END*. The control program then accepts the command word, validates it, and if it is acceptable will branch to the appropriate routine. After completion of the specified routine control passes back to the control program which requests another command word. The command words and their function are described below.

1. *NEWMODEL* As the command word indicates a new simulation model is to be set up. The program requests the names of all activities that are to be used in the model followed by the configuration parameters. The user is then required to supply the following: the number of transaction priority classes, number of transaction parameters, number of queues, facilities and storages, and the capacity of each storage. If the user wants to place transactions on the

future events list they are entered at this point, otherwise he can specify the values of *MEAN* and *MOD* to generate transactions automatically. Note that both methods may be used in the same model.

The program then requests the initial simulation time and the criteria for stopping the simulation run. The user can terminate the run in one of two ways: by specifying the length of time the model is to run or the number of transactions which are to be created and destroyed during the run. In the present program the user must supply values for both criteria; however, the effect of one can be eliminated by specifying a large value for it. The model is now assembled and control returns to the OSS control program which requests another command word.

One other important point should be mentioned in connection with this command word. Since OSS does not operate within the APL system directly, the task of setting up a model is at the present time "semi-automatic", i.e. the user must add activity names to parts of the OSS control programs while under direct control of the APL system. The areas of interest are shown in

Figure 5. The function *BRANCH* is used during a simulation run to call activities to perform the state changes. The array *ECNS* is used only during the assembly phase, i.e. when the *NEWMODEL* command word is specified. Note that the array contains the same activity names that appear in *BRANCH* and in the same order, and that the names must contain six characters including blanks if necessary. Then before a model can be assembled and run, the user must insert the names of all activities he will use in the function *BRANCH* and generate an array, *ECNS*, containing the same names that appear in *BRANCH*.

2. *STORE* This command allows a user to store all the required information of a model, currently active in the system, so that it may be retrieved and run at some later time. The user is required to supply a set of unique names by which various components of the model are stored.
3. *LOAD* This command allows a previously stored model to be retrieved and made current. The user must supply a set of names, as called for, by which the model has been previously stored.
4. *RUN* After a model has been established in memory (by either *NEWMODEL* or *LOAD* command words) this

```
      ∇BRANCH[□]∇

    ∇ R←BRANCH I
[1]   R←0
[2]   →(I=⍳14)/ 5 7 9 11 13 15 17 19 21 23 25 27 29
      31
[3]   TSI←10+R←1
[4]   →0
[5]   GENRAT
[6]   →0
[7]   MSGIN
[8]   →0                              FCNS
[9]   FACOUT
[10]  →0                          GENRAT
[11]  STORIN                      MSGIN
[12]  →0                          FACOUT
[13]  QTEST                       STORIN
[14]  →0                          QTEST
[15]  ADVANC                      ADVANC
[16]  →0                          READIS
[17]  READIS                      ENDISK
[18]  →0                          PROMES
[19]  ENDISK                      ENDMES
[20]  →0                          REJECT
[21]  PROMES                      MACHNC
[22]  →0                          ENDTRN
[23]  ENDMES
[24]  →0
[25]  REJECT
[26]  →0
[27]  MACHNC
[28]  →0
[29]  ENDTRN
[30]  →0
    ∇
```

FIGURE 5

command word will cause the model to be executed. It will continue until one of the limit criteria are exceeded, until the user presses the APL interrupt key, or until the APL interpreter is unable to continue because of an error. In the latter two cases control is taken over by the APL system and the user can examine any variables or arrays in the system, or make changes to any activities. The program can then be resumed at the point of interruption.

If the simulation runs to completion the control program prints the time at which the simulation ended and the number of transactions terminated. The user then has the option of obtaining a print-out of the statistics which were gathered during the run, after which another command word is requested.

5. *NEWRUN* Before executing a model this command allows the user to reinitialize some of the model parameters. For example, transactions may be placed on the future events list, *MEAN* and *MOD* may be specified and the length and initial value of simulated time may be specified.
6. *END* This command takes the user out of the OSS system and returns control to the APL system.

4.2.6 OSS Functions In addition to the transaction status statements described in Section 4.2.4.1 and the complete set of APL facilities, a number of additional statements have been implemented as APL functions to perform many operations required in simulation models. A description of each, and their syntax, follows.

1. *→(FACILITYIN A)/LABEL* This is used when an attempt is made to seize facility *A*. If the facility is occupied the current transaction is placed in a blocked state and the next statement in the activity to be executed is *LABEL*, where *LABEL* is a valid APL statement label. Otherwise the facility becomes occupied and the next sequential statement in the activity will be executed next. An alternate form, *R←FACILITYIN A*, where *R* is a dummy variable, may also be used. However, if the facility is occupied the next sequential statement will be executed next.
2. *FACILITYOUT A* Facility *A* will become available to another transaction. Corresponding statistics will be updated.
3. *→(A STORAGEIN B)/LABEL* This statement will add *B* units to the current contents of storage *A* unless the storage capacity will be exceeded.

If exceeded the current transaction is placed in a blocked state and the next executed statement is *LABEL*. Otherwise the storage is entered and the next sequential statement is executed next. The alternate form, *R←A STORAGEIN B* may also be used with the same result as described in 1. above.

4. *A STORAGEOUT B* *B* units are removed from storage *A* and the storage statistics are updated.
5. *A QUEUEIN B* *B* units are added to the current and total contents of queue *A*. Queue statistics are updated. The transactions are treated on a first-in first-out basis.
6. *A QUEUEOUT B* *B* units are removed from queue *A* and the relevant queue statistics are updated.
7. *A ASGN B* The value of parameter *A* of the current transaction becomes *B*, where *A* is less than or equal to the number of parameters specified in setting up the model, and *B* is any integer within the capacity of the machine.
8. *PARAMETER A* This will retrieve the value of the A^{th} parameter, not exceeding the specified number, of the current transaction.
9. *PRIORITY A* This will assign the current transaction a priority *A*.

10. *TRANSFER EXPRESSION* As described in Section 4.2.4.1 this statement is used in conjunction with a primary and secondary configuration parameter, and allows the programmer to change the path taken by various transactions through the model. The value of *EXPRESSION* determines whether the primary or secondary configuration parameter will be used as the next sequential activity; one for the primary and zero for the secondary configuration. Thus, *EXPRESSION* can be any valid logical APL expression. For example, *TRANSFER* (*QCONT* 1) ≤ 5, will cause the activity given by the primary configuration parameter to be executed next if the contents of queue 1 is less than six, otherwise the next activity will be given by the secondary parameter.
11. *FACILITY A* The result is the current state of facility *A*; 1 for occupied, 0 otherwise.
12. *STCONT A* The result is the current contents of storage *A*.
13. *QCONT A* The result is the current contents of queue *A*.
14. *A RAND B* This statement generates a number in the range $A \pm B$ ($B \leq A$), with equal probability given to each number in the range.

4.3 An Example

To illustrate the use of OSS a model will be constructed illustrating the operation of a simple telephone exchange. The system consists of a number of telephones, each connected to the exchange by its own line. The exchange contains a number of cross-links which can provide a connection between any two telephone lines entering the exchange, with only one connection being allowed to any given line at a time. It is assumed that calls which cannot be made at a designated time will be attempted until finally made. For simplicity assume the system contains 10 telephones and 5 cross-links.

4.3.1 Developing a Model To develop a model of the system the following are necessary:

1. Entities to indicate the begin and end time of each call and the two telephones involved in the call.
2. Entities to indicate the state of the telephones.
3. Entities to indicate the state of the cross-links.

The first entities can be recognized as temporary and will be represented as a transaction in the system, with the telephone indicators as transaction parameters.

Since the telephones and cross-links can be in only one of two states they can be represented as facilities.

Then before a call can be placed a check must be made if the telephones are available and if a cross-link is available. Queue statistics will also be obtained on those calls which cannot be placed at the earliest possible time.

The entities to be used are defined as follows:

Parameter 1 Telephone number sending call
Parameter 2 Telephone number receiving call
Parameter 3 Length of telephone call
Facility 1 Telephone number 1
⋮
Facility 10 Telephone number 10
Facility 11 Cross-Link number 1
⋮
Facility 15 Cross-Link number 5.

Telephone calls are placed every 10 ± 5 minutes, their length being 3 ± 2 minutes. One unit of simulated time will represent one minute of real time.

4.3.2 OSS Programs Excluding the activity to generate the transactions, three activities are required as shown in Appendix II. The following is a description of the function of each activity.

1. *MSGIN*

Line 1 & 2. Assign the telephone numbers used in this call to parameters one and two by use of the random number generator.

Line 3. Check to ensure different telephones are involved in the call, otherwise go to line 2 and reassign the telephone number receiving the call.

Line 4. Assign parameter three the length of the call chosen from the range 6 ± 2.

Line 5. Places this call into a queue of calls to be made causing the queue statistics to be updated.

2. *TELCAL*

Line 1. Checks the availability of both telephones required to initiate this call. If one is not available the call is placed in a blocked state at line 10.

Line 2. This line generates a vector *LINK* containing the facility numbers corresponding to available cross-links. If no cross-links are available the call is placed in a blocked state.

Line 3. Removes the call from the waiting queue.

Line 4 & 5. Sets the facilities corresponding to the required telephones to occupied.

Line 6. The first available corss-link is used to place the call.

Line 7. The number of the cross-link used is placed in transaction parameter four.

Line 8. The transaction is suspended an amount of time equal to the length of the call.

Line 9. Exit from the function.

Line 10. Places calls in a blocked state.

3. *ENDCAL*

Line 1 & 2. Releases the telephones used by this call.

Line 3. Releases the cross-link used by the call.

Line 4. Deletes the current transaction from the system.

4.3.3 Results After the activities described in the previous section were written and entered into the APL system, OSS was used to assemble them and run the resulting model. Appendix II contains a complete listing of the setting-up, running, and statistical output obtained from the model.

CHAPTER V

SOME APPLICATIONS OF OSS

In this chapter we will consider three systems which have been simulated using OSS. A description of each system will be given followed by a discussion of the simulation model developed for each system. Throughout this chapter STU will be used as an abbreviation for simulation time units and the real time corresponding to one STU will be specified for each problem.

5.1 Simulation of a Three Machine Assembly System

This problem is the same as that presented in Section 3.4 where it was simulated using the FORTRAN program. It has also been simulated using OSS to illustrate the differences in the two approaches from a programming point of view. Refer to Section 3.4.1 for a description of the problem and to Appendix III for a listing of the OSS activities used to simulated the problem. In this example one STU will represent one minute of real time.

Since the system contains components produced by machines A and B as temporary entities, these will be represented as transactions in the model. However, since they are produced and used in pairs, each trans-

action will represent two components, rather than one. Random numbers are used to determine if a component is to be rejected, i.e. an integer number in the range [1,10] is generated and if equal to 10 a component is rejected. Then, because one or both of the components of a transaction may be rejected, we must ensure that some of the transactions are terminated when a component is deleted. Three cases may be recognized.

(a) If both components are rejected the transaction corresponding to them is terminated.

(b) If neither component is rejected, the transaction is not terminated.

(c) If one component is rejected, terminate the transaction only if there is an even number of components in the storage bin, denoted in the model by *STORAGE* 1.

The number of components to be entered into the store by a transaction is indicated in transaction parameter 1. As shown in the first activity, *REJECT*, the first statement assigns parameter 1 a value of two. Subsequent statements in the activity decrease the parameter by one for every component rejected. The second half of the activity contains statements to determine the number of components to be entered in the storage

bin, to add that number, and determine whether the transaction should be terminated or not, using the above mentioned tests.

After a transaction has completed *REJECT* and has not been terminated it immediately attempts to seize *FACILITY* 1, representing machine C. This action is represented as the first statement in activity *MACHNC*. If the facility is occupied the transaction is placed in a blocked state and the activity is immediately exited. If the facility is not occupied it is seized by the current transaction, two components are removed from the storage bin, and the transaction is suspended for 15 STU, representing the processing time of machine C.

After 15 STU the suspended transaction is placed back on the current events list from the future events list and, when it becomes current, the last activity *ENDTRN* is executed. The state changes caused by this activity include incrementing the count of completed products, represented as *STORAGE* 2, by one and releasing facility one. The transaction is then deleted from the model.

The activities were assembled as shown in Appendix III. Note that in this and other problems which use the automatic generation of transactions the first activity specified is *GENRAT*, although no activity of that name

is actually executed. It is, however, necessary to allow the control program to set up the correct activity sequence, as well as providing the user with a listing indicating transactions were generated automatically.

The statistics obtained indicate 19 products were completed during a run simulating an eight-hour shift. During this time the maximum number of components in the storage bin was three and machine C was in operation 59.87% of the time.

5.2 Simulation of a Real-Time Data Processing System

The system being simulated in this section consists of a computer with three disk drives sharing a channel to the CPU, and a high speed on-line terminal. The computer receives messages from the terminal, and a search is made on one of the disk files to retrieve a record which is subsequently used by the computer for processing. Figure 6 illustrates the sequence of actions that are required in the model. In this example one STU is equivalent to one millisecond of real time.

Messages are sent by the terminal to the computer every 50 ± 25 STU at which time the CPU requires 1 STU to place the message in memory. Each message enters a queue since the disk file can search for only one record at a time. When the appropriate disk becomes available

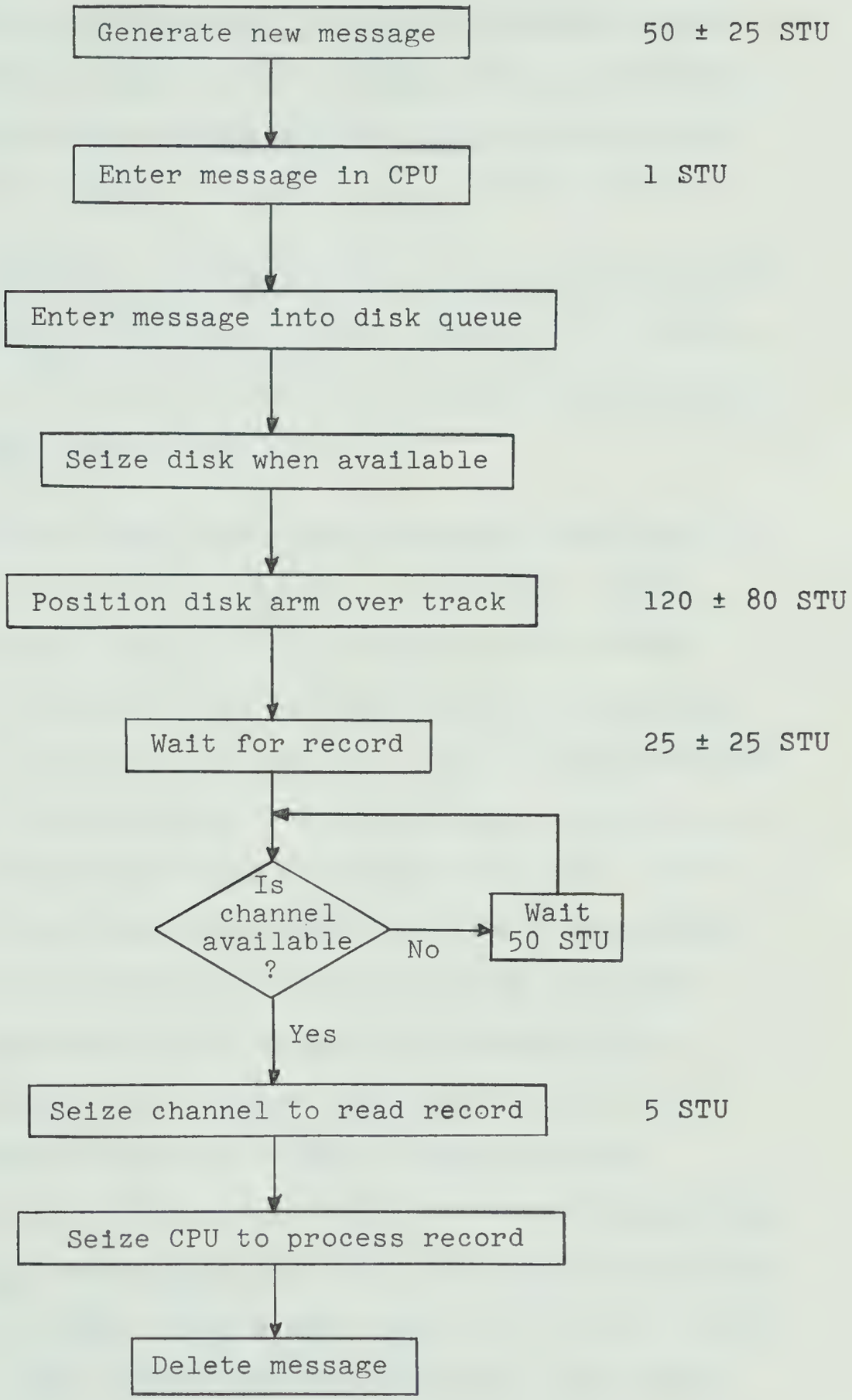
Generate new message
50 ± 25 STU
Enter message in CPU
1 STU
Enter message into disk queue
Seize disk when available
Position disk arm over track
120 ± 80 STU
Wait for record
25 ± 25 STU
Is channel available ?
No
Wait 50 STU
Yes
Seize channel to read record
5 STU
Seize CPU to process record
Delete message

FIGURE 6

the search request corresponding to the message at the top of the queue takes control of that disk. Before the correct record can be retrieved the following two conditions must be met.

(a) The disk arm must be positioned over the correct track of the disk. This requires 120 ± 80 STU.

(b) The correct record must be under the read head. This requires 25 ± 25 STU.

After satisfying these conditions the record may be read into the CPU if the channel is not being used by one of the other disks. If not available the attempt to read the record must be suspended until one complete revolution of the disk, or 50 STU. This is repeated until the channel is available. When the channel becomes available it is seized for 5 STU to perform the read. The record can then be processed when the CPU is available, after which the message is removed from the computer.

The temporary entity of OSS, the transaction, is used to represent the messages which are sent to the computer from the terminal. Each transaction has associated with it two parameters: the first indicating the length of the message, and the second indicates which of the three disks contains the record of interest. The transaction then, moving through the model, represents

the action of the equipment through time in response to a message being entered into the system. The system can be used to study the effect of different message inter-arrival times on the time required to process a given message due to the single channel.

The system equipment has been represented as the OSS entities listed as follows:

Facility	1	Disk unit 1
	2	Disk unit 2
	3	Disk unit 3
	4	Channel
	5	CPU
Storage	1	CPU storage for messages, capacity 1500 storage units.

In the example shown transactions are created every 50 ± 25 STU. The message length parameter is chosen from the rectangular distribution 15 ± 10. Associated with each disk is a queue, e.g. queue 1 with disk 1, which will be used to obtain statistics which can be used to judge the system performance. Following is a description of the activities, listed in Appendix III, used in the model.

1. *MSGIN* Each new transaction can enter the system only when the CPU is available. When available this activity assigns the two transaction para-

meters and suspends the transaction for 1 STU.

2. *FACOUT* This activity releases the CPU facility to another transaction.
3. *STORIN* If all the CPU storage is occupied by previous messages the transaction will be blocked until some storage is released. The activity puts the transaction in the queue indicated in parameter two, after being entered into the CPU storage.
4. *QTEST* If the disk facility is occupied the transaction remains in the queue. Otherwise it is removed from the queue and suspended for 120 ± 80 STU, representing the time required to position the read head over the correct track.
5. *ADVANC* The transaction is again suspended, but for 25 ± 25 STU, corresponding to waiting for the disk to rotate and position the correct record under the read head.
6. *READIS* After positioning the record in the previous two activities, this activity first checks for the availability of the channel. If not available the transaction is suspended for 50 STU when the activity is again attempted as indicated by the line *HOLD* 50. Otherwise the channel is occupied for 5 STU representing the reading of the record off the disk.

7. *ENDISK* This activity releases the disk and channel facilities to other transactions.
8. *PROMES* A check is then made on the CPU facility; if available the CPU is seized for a calculated time as the record is processed, otherwise the transaction is blocked.
9. *ENDMES* The CPU facility is released, the message is removed from CPU storage, and the transaction is terminated.

The results obtained for this example are shown in Appendix III. The run was completed after 2995 STU during which 47 messages had been completed.

5.3 Simulation of a Small Branch Library

This example is taken from Caras (1968) and is outlined in Figure 7. It is divided into two basic sections, books and documents.

People enter the library every 5 ± 2 STU and leave if there are more than eight persons waiting at the information desk. The people at the information desk are served on a first-come first-serve basis, each requiring 8 ± 3 STU, to be directed to either the books or documents section. 30% of the customers are sent to the books librarian, the remainder to the documents librarian.

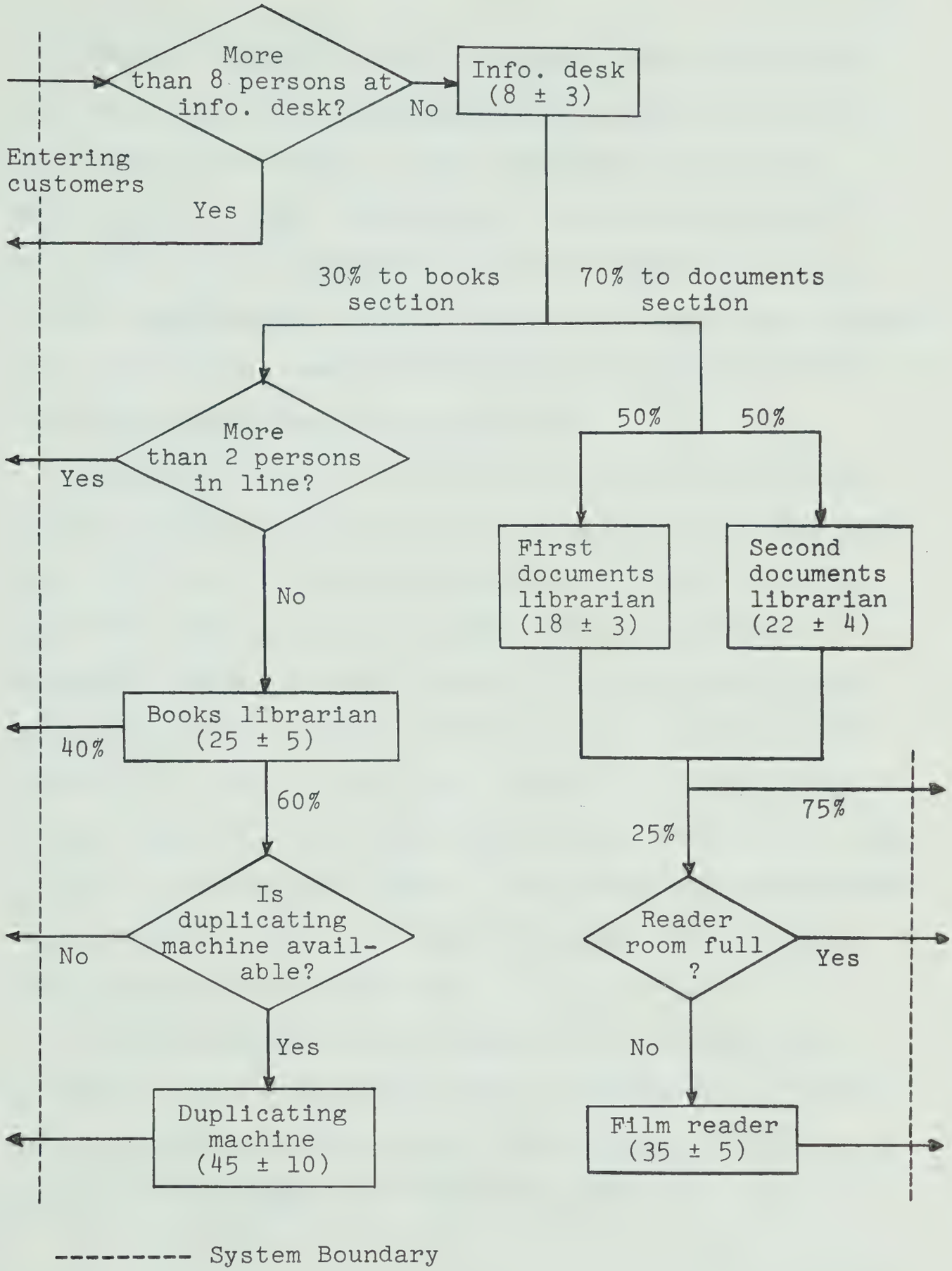
More than 8 persons at info. desk?
No
Info. desk (8 ± 3)
Entering customers
Yes
30% to books section
70% to documents section
More than 2 persons in line?
Yes
No
50%
50%
First documents librarian (18 ± 3)
Second documents librarian (22 ± 4)
Books librarian (25 ± 5)
40%
60%
75%
25%
Is duplicating machine available?
No
Yes
Reader room full ?
Yes
No
Duplicating machine (45 ± 10)
Film reader (35 ± 5)
System Boundary

FIGURE 7

There is only one books librarian, and if there are two people waiting for service the customers diverted to this section will leave. Otherwise they form a queue requiring 25 ± 5 STU for service. After being served 40% leave the system immediately while the remainder attempt to use a duplicating machine. If it is in use the customers leave the system, otherwise they use the machine for 45 ± 10 STU, after which they leave the system.

Customers diverted to the documents section divide evenly between two documents librarians forming queues at each. The first librarian requires 18 ± 3 STU to serve customers, the second 22 ± 4 STU. After leaving the documents librarian three-fourths of the customers leave the system. The remainder attempt to use a film reader which is in a small room with a capacity of three persons. If the room is full the customer leaves, otherwise he joins a queue for use of the reader. The person using the reader requires 35 ± 5 STU after which he leaves and the first person in the queue takes over.

Since we are obviously interested in the movement of people through the system each transaction will represent one person in the system. The various entities required in the model are defined as follows:

Facility	1	Information desk
	2	Books librarian
	3	Duplicating machine
	4	First documents librarian
	5	Second documents librarian
	6	Film reader
Storage	1	Film reader room, capacity 3 persons.

In addition each of the above facilities, except facility 3, may have lines of people queued up and waiting for service. To obtain information on the waiting times and number of persons waiting for each facility, a queue will be associated with each facility. The statistics obtained can be used to analyze the efficiency of each section and to pinpoint any bottlenecks in the system.

Following is a description of the activities required in this model, as listed in Appendix III.

1. *ENTER* The contents of the information desk queue is checked; if more than 8 people are waiting for service the customer leaves and the transaction is terminated. Otherwise the transaction is placed in queue 1.
2. *INFORM* This activity represents the action of the information desk. The first line checks the status of the facility. If available the trans-

action is removed from the queue and is suspended 8 ± 3 STU representing the time required by this customer at the information desk. Otherwise the transaction remains in the queue.

3. *SPLIT* This activity provides the necessary statements to accomplish three tasks. First a random number is used in conjunction with a *TRANSFER* statement to send 30% of the transactions to the books section and the remainder to the documents section. Second, the transactions routed to the books section check the contents of the waiting line at the books librarian. If there are no more than two people waiting for service the transaction is placed in the queue, otherwise the transaction is terminated indicating the customers leaving the system. Third, transactions routed to the documents section have a parameter assigned indicating which of the document librarians is to service the transactions.
4. *BOOKLI* If the book librarian is available the transaction is removed from the corresponding queue and suspended for 25 ± 5 STU, the service time of this librarian.
5. *DUPMAC* This activity terminates 40% of all transactions entering it, representing those

people who had no use of the duplicating machine. The remainder will also leave if the duplicating machine is not available. If the machine is available it will be engaged for 45 ± 10 STU.

6. *ENDTR2* Transactions leaving the duplicating machine release the facility and are terminated in this activity.
7. *DOCLIB* This is the path the majority of transactions will take after activity *SPLIT*. Each transaction carries in parameter 1 the queue in which it was placed in *SPLIT*, with queues 3 and 4 corresponding to the first and second documents librarian, respectively. The first line of this activity checks the availability of the required librarian and if available the transaction is suspended a period of time selected from the distribution corresponding to the required librarian.
8. *FILMQU* After releasing the document librarian facility three-fourths of the transactions are terminated. The remainder check the number of people waiting to use the film reader, and if greater than two, they also are terminated. Otherwise the transaction enters the film reader queue and storage entities.

9. *READER* A transaction finding the film reader (facility 6) available will be removed from the queue and suspended for 35 ± 5 STU representing the use of the reader.
10. *ENDTR1* The transaction leaving the film reader is removed from the storage and reader facility, and is terminated.

Figure 8 shows the manner in which the activities are joined together to form the model. The integers associated with each activity are those used as the configuration parameters when the model was being assembled. Note that for activity *SPLIT* the primary and secondary configuration parameters were specified as 5 and 8, respectively.

After 100 transactions had been generated and terminated, the results shown in Appendix III were obtained. It is apparent that in this hypothetical system facility 1, corresponding to the information desk, creates a bottleneck, since on the average people had to wait in line 67.81 time units before being served. Thus, better service at the information desk is necessary. On the other hand only 7 people used the film reader indicating another is not required.

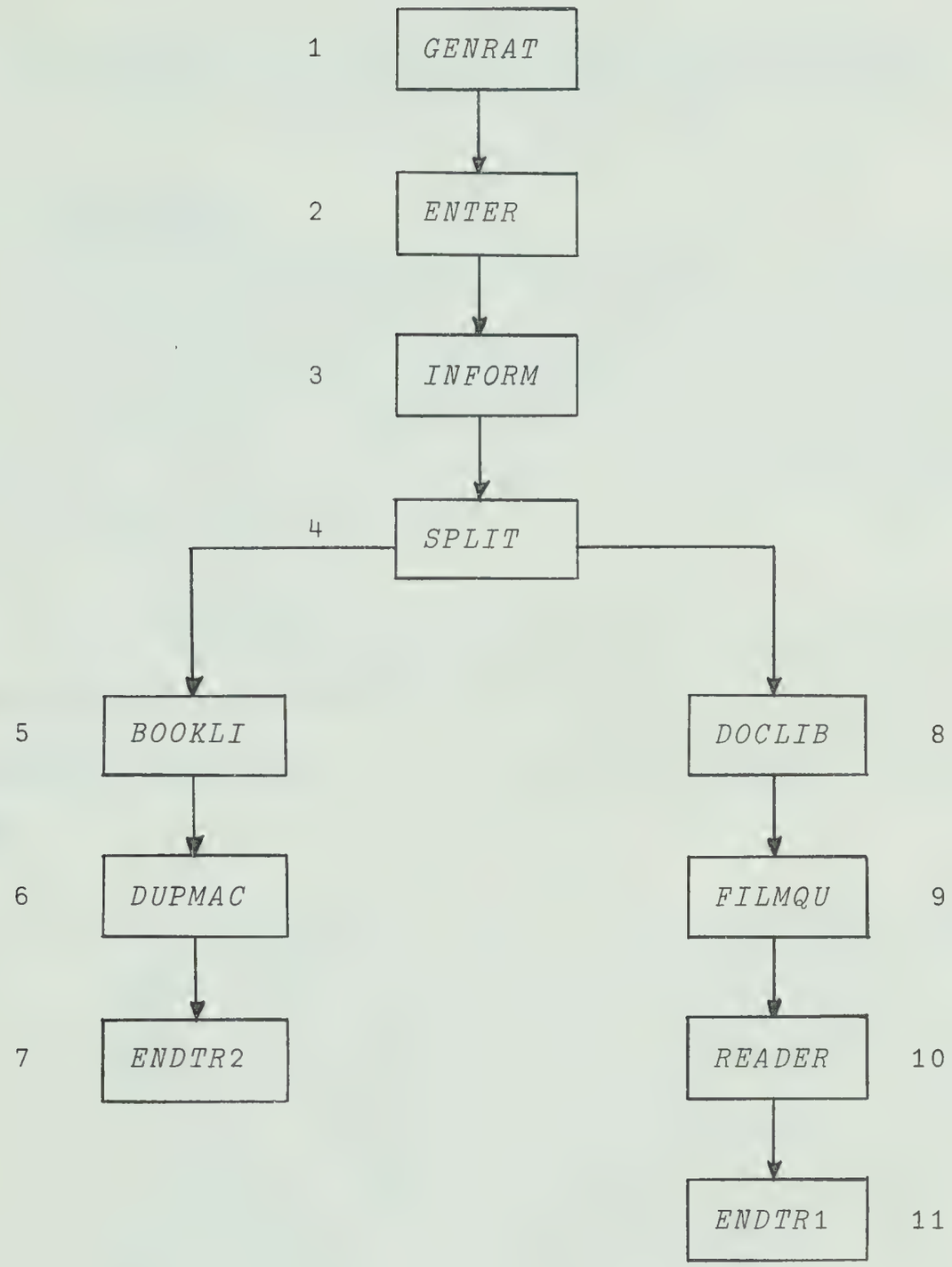
1
GENRAT
2
ENTER
3
INFORM
4
SPLIT
5
BOOKLI
6
DUPMAC
7
ENDTR2
8
DOCLIB
9
FILMQU
10
READER
11
ENDTR1

FIGURE 8

CHAPTER VI

CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

6.1 Conclusions

The following conclusions are based on a comparison of the two simulation programs presented in this thesis with the simulation language requirements specified in Sections 2.1, 2.2, and 4.1.2, and on experience gained in using the programs.

6.1.1 FORTRAN Simulation Program Although the program provided a majority of the requirements of a simulation language the following observations may be made.

1. The arrangement of all entities into one large array, with the same name for every type of entity, is awkward to use from a users' point of view. A better approach would be to provide a number of types of entities, e.g. facilities and storages, with each referenced by a name descriptive of that type.
2. Although a machine-based simulation language requires no algorithm to schedule the activities some mechanism must be provided to advance simulated time. The fixed time increment used in this

program is usually wasteful of computer time as activities will be attempted at times when no state changes will occur. This is a result of making the time increment small enough to ensure that all events will occur at their proper time. The use of special time entities such as described in Section 2.3.1.1 would overcome this disadvantage.

3. One essential feature of a simulation language, which this one is lacking, is the automatic generation of statistical or other performance data as the model is being run. This is a consequence of the entity structure mentioned in 1. above, since the user is free to define an entity to represent any required variable. It may be noted that a relatively large amount of data is obtained from the OSS simulation compared to that obtained from the FORTRAN simulation for approximately the same amount of programming effort.
4. Although a simulation language may provide all the features as indicated in Section 2.1 and 2.2, the utility of the language depends on how easily the given features can be used to construct a simulation model and on how realistic the final model may be. Thus, although the machine-based world view of this program permits the construction

of realistic models, many of the necessary features of a simulation language can only be obtained after considerable effort on the part of the user.

6.1.2 OSS

1. The OSS program provided all the simulation language features, including those of on-line systems, with the exception of a compiled mode of execution as this feature is not currently implemented in the APL system.
2. OSS proved easier to use than the FORTRAN program because:
   (a) it allowed the separation and referencing of entities by meaningful class names,
   (b) numerous statements were provided to perform operations on the entities, and
   (c) the simulation models were written, debugged, and assembled in a time-sharing environment.
3. As seen in the applications of OSS, the use of the event as the basic descriptive unit results in some activities that contain very few statements. This could be improved by incorporating the process as the descriptive unit. However, this implies modifying the APL system to permit "parallel" execution of the activities, in the

same manner modifications permitted SIMULA to be incorporated into ALGOL.

4. The powerful vector and array operators make APL an excellent language in which to write the OSS programs and the simulation models.
5. A detailed comparison of the two simulation programs presented in this thesis would be very difficult since they are written in different languages and operate in different environments. However, the following observations may be made. Of the two world views, material-based of OSS and machine-based of the FORTRAN program, the former was the easier in which to develop and write a simulation model. This can partly be attributed to the fact that some systems are more amenable to simulation by a material-based language than others. However, the deciding factor was the numerous utility functions, built-in statistics collecting functions and division of entities into classes available in OSS. In addition the ability to create and destroy temporary entities is a very useful feature in a simulation language.

6.2 Suggestions for Future Research

1. With the growing availability of on-line systems and the increasing complexity of simulation models, the need for an on-line simulation language will increase. Perhaps an attempt could be made to incorporate the OSS features directly into the APL system or as a subset of the language.
2. As mentioned in Section 6.1.2 the process represents a more desirable descriptive unit than the event currently used in OSS. Any plan to incorporate the OSS features into APL or any on-line language should include changing the descriptive unit to the process.
3. A relatively new piece of hardware used in time-sharing systems is the graphic display or CRT. Investigation could determine the usefulness of this input-output facility operating in conjunction with an on-line simulation language. The CRT could be applied to problems such as drawing flow diagrams, building activities, input of parameters, and output of simulation results either during or after simulation runs.

BIBLIOGRAPHY

Blunden, G.P. and Krasnow, H.S., 1967, "The Process Concept as a Basis for Simulation Modeling", Simulation, vol. 9, 2, pp. 83-93.

Brennan, R.D. and Linebarger, R.N., 1964, "A Survey of Digital Simulation: digital analog simulator programs", Simulation, vol. 3, 6, pp. 22-36.

Brennan, R.D. and Linebarger, R.N., 1965, "An Evaluation of Digital Analog Simulator Languages", Proc. IFIP Congress, vol. II, pp. 337-338.

Buxton, J.N. and Laski, J.G., 1962, "Control and Simulation Language", Comput. J., vol. 5, pp. 194-199.

Buxton, J.N., 1966, "Writing Simulations in C.S.L.", Comput. J., vol. 9, pp. 137-143.

Caras, G.J., 1968, "Computer Simulation of a Small Information System", Amer. Doc., vol. 19, 2, pp. 120-122.

Clancy, J.J. and Fineberg, M.S., 1965, "Digital Simulation Languages: A Critique and a Guide", Proc. AFIPS Fall Joint Comput Conf., vol. 27, pp. 23-36.

Dahl, O.-J. and Nygaard, K., 1966, "SIMULA - An ALGOL-Based Simulation Language", Comm. ACM, vol. 9, 9, pp. 671-678.

Dimsdale, B. and Markowitz, H.M., 1964, "A Description of the SIMSCRIPT Language", IBM Systems J., vol. 3, 1, pp. 57-67.

Evans, G.W., Wallace, G.F. and Sutherland, G.L., 1967, Simulation Using Digital Computers, Prentice-Hall, Inc., Englewood Cliffs, N.J.

Farmer, J.A. and Collcut, R.H., 1967, "Experience of Digital Simulations in a Large O.R. Group", in Digital Simulation in Operational Research, edited by S.H. Hollingdale, American Elsevier Publishing Company, Inc., New York, pp. 166-175.

Gordon, G., 1962, "A General Purpose Systems Simulator", IBM Systems J., vol. 1, pp. 18-32.

Greenberger, M., Jones, M.M., Morris, Jr., S.H. and Ness, D.N., 1965, On-Line Computation and Simulation: The OPS-3 System, The MIT Press, Cambridge, Mass.

Hammersly, J.M. and Handscomb, D.C., 1964, Monte Carlo Methods, Methuen & Co. Ltd., London.

Herscovitch, H. and Schneider, T.H., 1965, "GPSS III - An Expanded General Purpose Simulator", IBM Systems J., vol. 4, 3, pp. 174-183.

IBM, 1960, Job Shop Simulation Application, IBM Corp. Data Systems Div. M&A-1.

Iverson, K.E., 1962, A Programming Language, John Wiley & Sons, Inc., New York.

Jones, M.M., 1967a, "On-Line Simulation", Proc. - ACM National Mtg., pp. 591-599.

Jones, M.M., 1967b, "Incremental Simulation on a Time-Shared Computer", Unpublished Ph.D. Thesis, Alfred P. Sloan School of Management, MIT.

Kelley, D.H. and Buxton, J.N., 1962, "Montecode - An Interpretive Program for Monte Carlo Simulations", Comput. J., vol. 5, 2, pp. 88-93.

Kiviat, P.J., 1966, "Development of New Digital Simulation Languages", The J. of Ind. Eng., vol. XVII, 11, pp. 604-609.

Kiviat, P.J., 1967, "Development of Discrete Digital Simulation Languages", Simulation, vol. 8, 2, pp. 67-70.

Knuth, D.C. and McNeley, J.L., 1964, "SOL - A Symbolic Language for General-Purpose Systems Simulation", IEEE Trans. on Elec. Comput., vol. EC-13, 4, pp. 401-408.

Krasnow, H.S. and Merikallio, R.A., 1964, "The Past, Present and Future of General Simulation Languages", Management Sc., vol. 11, 2, pp. 236-267.

Krasnow, H.S., 1967, "Dynamic Presentation in Discrete Interaction Simulation Languages", in Digital Simulation in Operational Research, edited by S.H. Hollingdale, American Elsevier Publishing Company, Inc., New York, pp. 77-92.

Lewis, E.R., 1967, "Some Early Simulations", Simulation, vol. 8, 2, pp. V-VII.

Markowitz, H.M., Hausner, B. and Karr, H.W., 1963, SIMSCRIPT: A Simulation Programming Language, Prentice-Hall, Inc., Englewood Cliffs, N.J.

McNeley, J., 1967, "Simulation Languages", Simulation, vol. 9, 2, pp. 95-98.

Naur, P., 1963, "Revised Report on the Algorithmic Language ALGOL 60", Comm. ACM, vol. 6, 1, pp. 1-17.

Teichroew, D. and Lubin, J.F., 1966, "Computer Simulation - Discussion of the Technique and Comparison of Languages", Comm. ACM, vol. 9, 10, pp. 723-741.

Teichroew, D., Lubin, J.F. and Truitt, T.D., 1967, "Discussion of Computer Simulation Techniques and Comparison of Languages", Simulation, vol. 9, 4, pp. 181-190.

Tocher, K.D., 1965, "A Review of Simulation Languages", Oper. Res. Quart., vol. 16, 2, pp. 189-218.

Wallace, G.F., 1967, "The Character of Simulation", Simulation, vol. 8, 5, pp. 252-254.

APPENDIX I

LISTINGS OF FORTRAN SIMULATION PROGRAMS

This section contains a listing of the FORTRAN simulation programs, consisting of the mainline program and the subroutine INIT. Following these are listings of the subroutines described and used in the example of Section 3.4 and a listing of the output obtained when the example problem was run.

CONTROL PROGRAMS

```
C          GENERAL SIMULATION PROGRAM
C
C          NUSE IS INPUT UNIT OF USER
C          MON IS OUTPUT UNIT USED TO MONITOR USERS INPUT
C          NOUT IS OUTPUT TO USER
C          NDAT IS DATA INPUT UNIT DURING SIMULATION RUN
C          MAG IS MAGTAPE UNIT
C          ACTIVITY NAMES ARE 4 ALPHAMERIC CHARACTERS STORED
C          AS NAMOP(I),I=1,16
C          THE I-TH ACTIVITY OF ANY MODEL IS OF KIND(I) AND
C          NUMBER NUMB(I)
C          EACH MODEL IS STORED ON FOUR ADJACENT RECORDS
C          FINAL RECORD LISTS NO. DEFINED ACTIVITIES, NO.
C          DEFINED MODELS, AND NAMES OF DEFINED ACTIVITIES
C          NOPS=NUMBER OF DEFINED ACTIVITIES
C          NCON=NUMBER OF DEFINED MODELS
C
C          DIMENSION STATEMENTS
C
      DIMENSION NAMOP(16),KIND(144),NUMB(144),KPAR(100),
     1NKPAR(16),NPPAR(16),STATE(25,40),PPAR(100)
      DIMENSION IMCRD(2)
C
      COMMON/BLKA/STATE,PPAR,KPAR,NAMOP,NKPAR,NPPAR,KIND,NUMB
      COMMON/BLKB/NBLOK,NSTAT,NKP,NPP,KKP,KPP,INDEX
      COMMON/BLKC/NCON,NOPS,MAG,NUSE,MON,NOUT,NDAT,TIME
C
C          (1)  SPECIFY UNIT NUMBERS AND COMMAND CODES
C
      NUSE=5
      MON=6
      NOUT=6
      NDAT=5
      MAG=3
      DATA KODE1,KODE2,KODE3,KODE4,KODE5,KODE6,KODE7 / 'RUN ',
     1'CALL', 'STOR', 'NEWA', 'NEWM', 'NEWT', '    ' /
      DATA INITAL/ 'INIT' / , ND/'END '/
C
C          SET NOSTO = -1 TO INDICATE NO MODEL IN STORAGE
C
      NOSTO=-1
      NEG=-1
C
C          SET NCON=-1 TO INDICATE MAGTAPE HEAD MUST BE SET
C
```

```
      NCON=-1
      WRITE(NOUT,101)
101   FORMAT('1')
C
C          (2)  REQUEST A BASIC COMMAND
C
200   WRITE(NOUT,210)
210   FORMAT('  C) ENTER RUN/CALL/STORE/NEWACT/NEWMOD/' ,
     1'NEWTAPE/END/INIT/')
      READ(NUSE,220) IWORD
220   FORMAT(2A4)
      WRITE(MON,230) IWORD
230   FORMAT('  U) ',2A4)
      ITEM = IWORD(1)
      IF(ITEM.EQ.ND) STOP
      IF(ITEM.EQ.KODE6) GO TO 6000
C
C          POSITION READ HEAD AT END OF MAGTAPE IF NOT DONE
C          PREVIOUSLY
C
233   IF(NCON.NE.(-1))GO TO 250
      REWIND MAG
240   READ(MAG) I
      IF(I.NE.(-1))GO TO 240
245   BACKSPACE MAG
      READ(MAG) NEG,NOPS,NCON,NAMOP
C
C          BRANCH TO APPROPRIATE ROUTINE DETERMINED BY BASIC
C          COMMAND
C
250   IF(ITEM.EQ.KODE1) GO TO 1000
      IF(ITEM.EQ.KODE2) GO TO 2000
      IF(ITEM.EQ.KODE3) GO TO 3000
      IF(ITEM.EQ.KODE4) GO TO 4000
      IF(ITEM.EQ.KODE5) GO TO 5000
      IF(ITEM.EQ.INITAL) CALL INIT
      GO TO 200
C
C          (3)  USER HAS ENTERED NEWTAPE
C               PLACE SINGLE LOGICAL RECORD ON MAGTAPE
C               (-1,0,0,16 BLANK WORDS)
C
6000  REWIND MAG
      NOPS=0
      NCON=0
      DO 6010 J=1,16
6010  NAMOP(J)=KODE7
      WRITE(MAG) NEG,NOPS,NCON,NAMOP
      GO TO 200
```

```
C
C          (4)  USER HAS ENTERED NEWACT
C               REQUEST NAMES, COUNT THEM, STORE IN LAST
C               RECORD OF MAGTAPE
C
4000  BACKSPACE MAG
      WRITE(NOUT,4010)
4010  FORMAT('  C) ENTER 4-CHARACTER NAMES OF NEW ACTIVITIES')
      WRITE(NOUT,4020)
4020  FORMAT('  C) EACH FOLLOWED BY A SINGLE BLANK')
      ITEM=NOPS+1
      READ(NUSE,4030) (NAMOP(J),J=ITEM,16)
4030  FORMAT(16(A4,1X))
      WRITE(MON,4040) (NAMOP(J),J=ITEM,16)
4040  FORMAT('  U) ',16(A4,1X))
      WRITE(NOUT,4045)
4045  FORMAT('  C) BEFORE DEFINING THEIR CONFIGURATION,COMPILE')
      DO 4070 J=ITEM,16
      IF(NAMOP(J).EQ.KODE7) GO TO 4072
      NOPS=NOPS+1
      WRITE(NOUT,4060) J,NAMOP(J)
4060  FORMAT('  C) SUBROUTINE OPN',I2,' FOR ACTIVITY ',A4)
4070  CONTINUE
4072  WRITE(MAG) NEG,NOPS,NCON,NAMOP
      GO TO 200
C
C          (5)  USER HAS ENTERED NEWMOD
C               REQUEST NSTAT = NO. OF STATE ENTITIES
C               REQUEST NBLOK = NO. OF ENTITY BLOCKS
C               REQUEST NTOT = NO. OF ACTIVITIES
C               REQUEST SEQUENCE OF OPERATIONS
C               SET TIME = 0.0
C
5000  WRITE(NOUT,5010)
5010  FORMAT('  C) SPECIFY NUMBER OF STATE ENTITIES AND ',
     1'BLOCKS IN FORMAT (2I2)')
      READ(NUSE,5020) NSTAT,NBLOK
5020  FORMAT(2I2)
      WRITE(MON,5030) NSTAT,NBLOK
5030  FORMAT('  U) ',2I2)
      TIME = 0.0
      WRITE(NOUT,5053)
5053  FORMAT('  C) ENTER TOTAL NO. ACTIVITIES IN MODEL IN',
     1' FORMAT (I5)')
      READ(NUSE,5036) NTOT
5036  FORMAT(I5)
      WRITE(MON,5037) NTOT
5037  FORMAT('  U) ',I5)
      WRITE(NOUT,5040)
5040  FORMAT('  C) LIST SEQUENCE OF ACTIVITIES')
```

```
      READ(NUSE,5050) ((KIND(J),NUMB(J)),J=1,NTOT)
5050  FORMAT(13(A4,I1,1X))
      WRITE(MON,5055) ((KIND(J),NUMB(J)),J=1,NTOT)
5055  FORMAT('  O) ',13(A4,I1,1X))
C
C         COMPARE LISTED NAMES WITH NAMOP(J) FOR J=1,NOPS
C         AND SET KIND(J), NUMB(J) FOR J = 1,NTOT
C
      DO 5080 J = 1,NTOT
      IF(KIND(J).EQ.KODE7) GO TO 5100
      DO 5060 I=1,NOPS
5058  IF(KIND(J).EQ.NAMOP(I)) GO TO 5070
5060  CONTINUE
      WRITE(NOUT,5065) KIND(J)
5065  FORMAT('  C) ',A4,' IS NOT A DEFINED ACTIVITY')
      GO TO 200
5070  KIND(J)=I
5080  CONTINUE
C
C         CALL SUBROUTINES TO DETERMINE THE NUMBER OF
C         CONFIGURATION AND ACTIVITY PARAMETERS FOR EACH
C         ACTIVITY OF THE MODEL
C
5100  INDEX=1
      DO 5140 I=1,NTOT
      ITEM=KIND(I)
      GO TO(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16),ITEM
5120  NKPAR(I)=NKP
      NPPAR(I)=NPP
5140  CONTINUE
C
C         REQUEST CONFIGURATION PARAMETERS KPAR(J)
C
      WRITE(NOUT,5150)
5150  FORMAT('  C) CONFIGURATION PARAMETERS IN FORMAT(25I3)')
      J1=1
      DO 5190 I=1,NTOT
      ITEM=KIND(I)
      WRITE(NOUT,5160) NKPAR(I),NAMOP(ITEM),NUMB(I)
5160  FORMAT('  C) LIST ',I2,' FOR ',A4,I1)
      IF(NKPAR(I).EQ.0) GO TO 5190
      J2=J1-1+NKPAR(I)
      IF(J2.LE.100) GO TO 5168
5164  WRITE(NOUT,5165)
5165  FORMAT('  C) MORE THAN 100 CONFIGURATION PARAMETERS')
      GO TO 200
5168  READ(NUSE,5170) (KPAR(J),J=J1,J2)
5170  FORMAT(25I3)
      WRITE(MON,5180)(KPAR(J),J=J1,J2)
5180  FORMAT('  O) ',25I3)
```

```
      J1=J2+1
5190  CONTINUE
C
C          REQUEST PROCESS PARAMETERS PPAR(J)
C
      WRITE(NOUT,5250)
5250  FORMAT('  C) ACTIVITY PARAMETERS IN FORMAT (4F20.10)')
      J1=1
      DO 5290 I=1,NTOT
      ITEM=KIND(I)
      WRITE(NOUT,5260) NPPAR(I),NAMOP(ITEM),NUMB(I)
5260  FORMAT('  C) LIST ',I2,' FOR ',A4,I1)
      IF(NPPAR(I).EQ.0) GO TO 5290
      J3=J1-1+NPPAR(I)
      IF(J3.LE.100) GO TO 5268
5264  WRITE(NOUT,5265)
5265  FORMAT('  C) MORE THAN 100 PROCESS PARAMETERS')
      GO TO 200
5268  READ(NUSE,5270) (PPAR(J),J=J1,J3)
5270  FORMAT(4F20.10)
      WRITE(MON,5280) (PPAR(J),J=J1,J3)
5280  FORMAT('  C) ',4F20.10)
      J1=J3+1
5290  CONTINUE
      NOST=1
      GO TO 200
C
C          (6)  USER HAS ENTERED STORE
C               REPLACE HEADER RECORD BY FOUR RECORDS FOR A
C               MODEL AND THE UPDATED HEADER RECORD
C               FIRST CHECK THAT A MODEL IS AVAILABLE
C
3000  IF(NUSTO.GT.0) GO TO 3030
3010  WRITE(NOUT,3020)
3020  FORMAT('  C) NO MODEL TO STORE')
      GO TO 200
3030  BACKSPACE MAG
      WRITE(MAG) J2,J3,NTOT,TIME,NSTAT,NBLOK,(KIND(I),I=1,NTOT)
      WRITE(MAG) (KPAR(I),I=1,J2),(PPAR(J),J=1,J3)
      WRITE(MAG) (NKPAR(I),I=1,NTOT),(NPPAR(I),I=1,NTOT)
      WRITE(MAG) ((STATE(I,J),J=1,NSTAT),I=1,NBLOK)
      NCON=NCON+1
      WRITE(MAG) NEG,NOPS,NCON,NAMCP
      WRITE(NOUT,3040) NCON
3040  FORMAT('  C) STORED ON TAPE AS MODEL NO. ',I5)
      GO TO 200
```

```
C
C          (7)  USER HAS ENTERED CALL
C               RETRIEVE A MODEL FROM MAGNETIC TAPE
C
2000  WRITE(NOUT,2010)
2010  FORMAT('  C) ENTER MODEL NUMBER IN FORMAT(I5)')
      READ(NUSE,2020) ITEM
2020  FORMAT(I5)
      WRITE(MON,2030) ITEM
2030  FORMAT('  J) ',I5)
      IF(NCON.GE.ITEM) GO TO 2060
      WRITE(NOUT,2050) NCON
2050  FORMAT('  C) ONLY',I5,' MODELS ON TAPE')
      GO TO 200
2060  J=3*(NCON-ITEM)+4
C
C          BACKSPACE J RECORDS
C
2070  BACKSPACE MAG
      J=J-1
      IF(J) 2080,2080,2070
2080  READ(MAG) J2,J3,NTOT,TIME,NSTAT,NBLOK,(KIND(I),I=1,NTOT)
      READ(MAG) (KPAR(I),I=1,J2),(PPAR(II),II=1,J3)
      READ(MAG) (NKPAR(I),I=1,NTOT),(NPPAR(I),I=1,NTOT)
      READ(MAG) ((STATE(I,J),J=1,NSTAT),I=1,NBLOK)
      NOSTO=1
      J=3*(NCON-ITEM)
C
C          FORWARD MAGTAPE J RECORDS
C
      IF(J) 2095,2095,2090
2090  READ(MAG)
      J=J-1
      IF(J) 2095,2095,2090
2095  READ(MAG) NEG,NOPS,NCON,NAMUP
      GO TO 200
C
C          (8)  USER HAS ENTERED RUN
C               REQUEST TIME INCREMENT AND DURATION OF RUN
C               IN FORMAT (2F20.10)
C               KKP AND KPP ARE FOR TEMPORARY COUNT OF
C               CONFIGURATION(KPAR) AND ACTIVITY(PPAR)
C               PARAMETERS, RESPECTIVELY
C
1000  WRITE(NOUT,1010)
1010  FORMAT('  C) ENTER TIME INCREMENT AND DURATION IN ' ,
     1'FORMAT (2F20.10)')
      READ(NUSE,1020) DEL,DUR
1020  FORMAT(2F20.10)
```

```
      WRITE(MUN,1030) DEL,DUR
1030  FORMAT('  U) ',2F20.10)
      ITER=DUR/DEL + 1
      INDEX=2
      DO 1090 NC=1,ITER
      KKP=1
      KPP=1
      DO 1060 I=1,NTOT
      ITEM=KIND(I)
C
C        TO EXECUTE ACTIVITY KIND(I)
C
      GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16),ITEM
1050  KKP=KKP+NKPAR(I)
      KPP=KPP+NPPAR(I)
1060  CONTINUE
      CALL OUTPUT
      TIME=TIME+DEL
1090  CONTINUE
      GO TO 200
C
C              BRANCH TO SUBROUTINES
C
1     CALL OPN1
      GO TO (5120,1050),INDEX
2     CALL OPN2
      GO TO (5120,1050),INDEX
3     CALL OPN3
      GO TO (5120,1050),INDEX
4     CALL OPN4
      GO TO (5120,1050),INDEX
5     CALL OPN5
      GO TO (5120,1050),INDEX
6     CALL OPN6
      GO TO (5120,1050),INDEX
7     CALL OPN7
      GO TO (5120,1050),INDEX
8     CALL OPN8
      GO TO (5120,1050),INDEX
9     CALL OPN9
      GO TO (5120,1050),INDEX
10    CALL OPN10
      GO TO (5120,1050),INDEX
11    CALL OPN11
      GO TO (5120,1050),INDEX
12    CALL OPN12
      GO TO (5120,1050),INDEX
13    CALL OPN13
      GO TO (5120,1050),INDEX
```

```
14      CALL OPN14
        GO TO (5120,1050),INDEX
15      CALL OPN15
        GO TO (5120,1050),INDEX
16      CALL OPN16
        GO TO (5120,1050),INDEX
        STOP
        END
```

```
      SUBROUTINE INIT
C
C              SUBROUTINE TO INITIALIZE STATE ENTITIES
C
      COMMON/BLKA/STATE,PPAR,KPAR,NAMOP,NKPAR,NPPAR,KIND,NUMB
      COMMON/BLKB/NBLOK,NSTAT,NKP,NPP,KKP,KPP,INDEX
      COMMON/BLKC/NCON,NOPS,MAC,NUSE,MON,NOUT,NDAT,TIME
      DIMENSION NAMOP(16),KIND(144),NUMB(144),KPAR(100),
     1NKPAR(16),NPPAR(16),STATE(25,40),PPAR(100)
      DATA KYES/'YES '/
C
C          IF NCLR = 'YES' MATRIX STATE IS CLEARED TO ZERO
C
      WRITE(MON,65)
65    FORMAT('  C) ENTER YES IF ALL STATE ENTITIES ARE',
     1' TO BE SET TO ZERO')
      READ(NUSE,68) NCLR
68    FORMAT(A4)
      WRITE(MON,70) NCLR
70    FORMAT('  U) ',A4)
      IF(NCLR.NE.KYES) GO TO 300
      DO 200 I=1,25
      DO 199 J=1,40
100   STATE(I,J)=0.0
199   CONTINUE
200   CONTINUE
      WRITE(MON,202)
202   FORMAT('  C) ALL STATE ENTITIES ARE ZERO')
C
C          IF NCLR = 'YES' SOME ENTITIES ARE SET TO
C          NON-ZERO VALUES
C
300   WRITE(MON,310)
310   FORMAT('  C) ENTER YES IF SOME BLOCKS TO BE RESET')
      READ(NUSE,68)NCLR
      WRITE(MON,70)NCLR
      IF(NCLR.NE.KYES) RETURN
      WRITE(MON,350)
350   FORMAT('  C) ENTER BLOCK NUMBER, INDEX, VALUE IN ',
     1'FORMAT (2I3,F12.5)' / '  C) BLOCK NUMBER OF ZERO',
     2' INDICATES LAST ')
355   READ(NUSE,360)IT,IT1,VAL
360   FORMAT(2I3,F12.5)
      WRITE(MON,365)IT,IT1,VAL
365   FORMAT('  U) ',2I3,F12.5)
      IF(IT.EQ.0) RETURN
      STATE(IT,IT1)=VAL
      GO TO 355
      END
```

EXAMPLE - SECTION 3.4

```
      SUBROUTINE OPN1
C
C   IF STATE(I,J)=1 ADD 5 TO STATE(I,K)
C   I IS THE BLOCK ENTITY NUMBER
C   J IS THE LOCATION OF THE FLAG
C   K IS THE ENTITY TO BE ADDED TO
C
      COMMON/BLKA/STATE,PPAR,KPAR,NAMOP,NKPAR,NPPAR,KIND,NUMB
      COMMON/BLKB/NBLOK,NSTAT,NKP,NPP,KKP,KPP,INDEX
      COMMON/BLKC/NCON,NCPS,MAG,NUSE,MON,NOUT,NDAT,TIME
      DIMENSION NAMOP(16),KIND(144),NUMB(144),KPAR(100),
     1NKPAR(16),NPPAR(16),STATE(25,40),PPAR(100)
      NKP=3
      NPP=0
      GO TO (1,2),INDEX
1     RETURN
2     I = KPAR(KKP)
      J = KPAR(KKP + 1)
      K = KPAR(KKP + 2)
      IF(STATE(I,J).EQ.0.0) RETURN
      STATE(I,K) = STATE(I,K) + 5.0
      RETURN
      END
```

```
      SUBROUTINE UPN2
C
C   IF STATE(1,2) = 20 SET STATE(1,2) = 0
C   GENERATE TWO RANDOM NUMBERS - IF LESS THAN
C   0.9 SET STATE(2,I) = 1,I=1,2
C   ADD ONE TO STATE(2,3) FOR EVERY REJECT
C
      COMMON/BLKA/STATE,PPAR,KPAR,NAMOP,NKPAR,NPPAR,KIND,NUMB
      COMMON/BLKB/NBLOK,NSTAT,NKP,NPP,KKP,KPP,INDEX
      COMMON/BLKC/NCON,NCPS,MAC,NOSE,MON,NOUT,NDAT,TIME
      DIMENSION NAMOP(16),KIND(144),NUMB(144),KPAR(100),
     1NKPAR(16),NPPAR(16),STATE(25,40),PPAR(100)
      NKP = 0
      NPP = 0
      IF(INDEX.EQ.1) IX = 13579
      GO TO (1,2),INDEX
1     RETURN
2     IF(STATE(1,2).LT.20.0) RETURN
      STATE(1,2) = 0.0
      DO 10 I = 1,2
      CALL RANDU(IX,IY,YFL)
      IX = IY
      IF(YFL.LE.0.9) GO TO 5
      STATE(2,3) = STATE(2,3) + 1.0
      GO TO 10
5     STATE(2,I) = 1.0
10    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE OPN3
C
C  ADDS STATE(2,1) AND STATE(2,2) TO STATE(3,1)
C  SETS STATE(2,1) AND STATE(2,2) TO ZERO
C
      COMMON/BLKA/STATE,PPAR,KPAR,NAMOP,NKPAR,NPPAR,KIND,NUMB
      COMMON/BLKB/NBLOK,NSTAT,NKP,NPP,KKP,KPP,INDEX
      COMMON/BLKC/NCON,NCPS,MAG,NUSE,MON,NOUT,NDAT,TIME
      DIMENSION NAMOP(16),KIND(144),NUMB(144),KPAR(100),
     1NKPAR(16),NPPAR(16),STATE(25,40),PPAR(100)
      NKP = 0
      NPP = 0
      GO TO (1,2),INDEX
1     RETURN
2     STATE(3,1) = STATE(3,1) + STATE(2,1) + STATE(2,2)
      STATE(2,1) = 0.0
      STATE(2,2) = 0.0
      RETURN
      END
```

```
      SUBROUTINE OPN4
C
C   STATE(3,3) = 1 INDICATES MACHINE C BUSY
C   IF BUSY STATE(3,2) INDICATES THE TIME
C   USED IN THIS CYCLE - IF = 15 END CYCLE
C   IF NOT BUSY CHECK STATE(3,1) FOR TOTAL PARTS
C   AVAILABLE -  IF GE 2 SET MACHINE C TO BUSY
C
      COMMON/BLKA/STATE,PPAR,KPAR,NAMOP,NKPAR,NPPAR,KIND,NUMB
      COMMON/BLKB/NBLOK,NSTAT,NKP,NPP,KKP,KPP,INDEX
      COMMON/BLKC/NCON,NOPS,MAC,NUSE,MCN,NOUT,NDAT,TIME
      DIMENSION NAMOP(16),KIND(144),NUMB(144),KPAR(100),
     1NKPAR(16),NPPAR(16),STATE(25,40),PPAR(100)
      NKP = 0
      NPP = 0
      GO TO (1,2),INDEX
1     RETURN
2     IF(STATE(3,2).NE.1.0) GO TO 10
      STATE(4,2) = STATE(4,2) + 5.0
      IF(STATE(3,3).LT.15.0) RETURN
      STATE(3,3) = 0.0
      STATE(3,2) = 0.0
      STATE(4,1) = STATE(4,1) + 1.0
10    IF(STATE(3,1).LT.2.0) RETURN
      STATE(3,1) = STATE(3,1) - 2.0
      STATE(3,2) = 1.0
      RETURN
      END
```

```
      SUBROUTINE OUTPUT
C
C   THIS SUBROUTINE IS USED TO OBTAIN
C   RESULTS FROM A SIMULATION RUN
C   IT MUST BE WRITTEN FOR EACH MODEL
C
C   THIS SUBROUTINE PRODUCES OUTPUT FOR
C   THE EXAMPLE PROBLEM OF CHAPTER 3
C
      COMMON/BLKA/STATE,PPAR,KPAR,NAMOP,NKPAR,NPPAR,KIND,NUMB
      COMMON/BLKB/NBLOK,NSTAT,NKP,NPP,KKP,KPP,INDEX
      COMMON/BLKC/NCON,NOPS,MAG,NUSE,MON,NOUT,NDAT,TIME
      DIMENSION NAMOP(16),KIND(144),NUMB(144),KPAR(100),
     1NKPAR(16),NPPAR(16),STATE(25,40),PPAR(100)
      IF(TIME.LT.480.0) RETURN
      WRITE(6,10) STATE(4,1),STATE(4,2),STATE(2,3)
10    FORMAT(' ','TOTAL OUTPUT OF MACHINE 3',F10.1, /
     1' ','TOTAL TIME MACHINE 3 RUNNING',F10.1, /
     2' ','TOTAL NUMBER REJECTS',F10.1)
      RETURN
      END
```

RESULTS

```
C) ENTER RUN/CALL/STORE/NEWACT/NEWMOD/NEWTAPE/INIT/
U) NEWTAPE
C) ENTER RUN/CALL/STORE/NEWACT/NEWMOD/NEWTAPE/INIT/
U) NEWACT
C) ENTER 4-CHARACTER NAMES OF NEW ACTIVITIES
C) EACH FOLLOWED BY A SINGLE BLANK
U) SETF RAND STOR MACH
C) BEFORE DEFINING THEIR CONFIGURATION,COMPILE
C) SUBROUTINE OPN 1 FOR ACTIVITY SETF
C) SUBROUTINE OPN 2 FOR ACTIVITY RAND
C) SUBROUTINE OPN 3 FOR ACTIVITY STOR
C) SUBROUTINE OPN 4 FOR ACTIVITY MACH
C) ENTER RUN/CALL/STORE/NEWACT/NEWMOD/NEWTAPE/INIT/
U) NEWMOD
C) SPECIFY NUMBER OF STATE ENTITIES AND BLOCKS IN FORMAT (2I2)
U)  3 4
C) ENTER TOTAL NO. ACTIVITIES IN MODEL IN FORMAT (I5)
U)     5
C) LIST SEQUENCE OF ACTIVITIES
U) SETF1 RAND1 STOR1 MACH1 SETF2
C) CONFIGURATION PARAMETERS IN FORMAT (20I5)
C) LIST  3 FOR SETF1
U)   1  1  2
C) LIST  0 FOR RAND1
C) LIST  0 FOR STOR1
C) LIST  0 FOR MACH1
C) LIST  3 FOR SETF2
U)   3  2  3
C) ACTIVITY PARAMETERS IN FORMAT (4F20.10)
C) LIST  0 FOR SETF1
C) LIST  0 FOR RAND1
C) LIST  0 FOR STOR1
C) LIST  0 FOR MACH1
C) LIST  0 FOR SETF2
C) ENTER RUN/CALL/STORE/NEWACT/NEWMOD/NEWTAPE/INIT/
U) INIT
C) ENTER YES IF ALL STATE ENTITIES ARE TO BE SET TO ZERO
U) YES
C) ALL STATE ENTITIES ARE ZERO
C) ENTER YES IF SOME BLOCKS TO BE RESET
U) YES
C) ENTER BLOCK NUMBER, INDEX, VALUE IN FORMAT (2I3,F12.5)
C) BLOCK NUMBER OF ZERO INDICATES LAST
U)   1  1      1.00000
U)   0  0      0.0
```

```
C) ENTER RUN/CALL/STORE/NEWACT/NEWMOD/NEWTAPE/INIT/
U) RUN
C) ENTER TIME INCREMENT AND DURATION IN FORMAT (2F20.10)
U)          5.0000000000       480.0000000000
TOTAL OUTPUT OF MACHINE C       19.0
TOTAL TIME MACHINE C RUNNING      290.0
TOTAL NUMBER REJECTS        8.0
C) ENTER RUN/CALL/STORE/NEWACT/NEWMOD/NEWTAPE/INIT/
U) END
```

APPENDIX II

LISTINGS OF OSS PROGRAMS

This section contains listings of all the APL programs required by OSS. They have been divided into the following sections.

1. Control Programs The function *SIMDRIV* accepts the command words and branches to the appropriate routine. The other functions in this section are called directly or indirectly by *SIMDRIV* to perform the functions indicated by the command words.
2. Scheduling Programs The programs of this section are used only when a simulation model is being run.
3. Transaction Status Programs These programs set the global variable *TSI* which indicates the state of the current transaction.
4. Utility Programs These programs are the OSS functions described in Section 4.2.6.

Following the listings of these programs is a listing of the activities and results for the example of Section 4.3.

CONTROL PROGRAMS

```
      ∇OSS[□]∇

    ∇ OSS
[1]   SIMDRIV
    ∇
```

```
      ∇SIMDRIV[□]∇

    ∇ SIMDRIV
[1]   DRIV1:'ENTER COMMAND WORD',□←' '
[2]   T←6ρ(T←INP⍞),6ρ' '
[3]   ' '
[4]   →(∧/T[⍳3]='RUN')/DRIV10
[5]   →(∧/T[⍳4]='NEWR')/DRIV20
[6]   →(∧/T[⍳4]='LOAD')/DRIV30
[7]   →(∧/T[⍳4]='STOR')/DRIV40
[8]   →(∧/T[⍳4]='NEWM')/DRIV50
[9]   →(∧/T[⍳3]='END')/DRIV70
[10]  →DRIV1,ρ□←'INVALID COMMAND WORD'
[11]  DRIV10:RUN
[12]  →DRIV1
[13]  DRIV20:INIT
[14]  →DRIV10
[15]  DRIV30:LDST 'LOAD'
[16]  →DRIV1
[17]  DRIV40:LDST 'STORE'
[18]  →DRIV1
[19]  DRIV50:NEWMODEL
[20]  →DRIV1
[21]  DRIV70:'SIMULATION COMPLETED'
    ∇
```

```
      ∇INP[□]∇

    ∇ R←INP X
[1]   R←((X≠' ')/X)
    ∇
```

```
      ∇RUN[□]∇

    ∇ RUN;R
[1]   R←SCAN
[2]   →R/6
[3]   'TRANSACTION STATUS ERROR'
[4]   ('STATUS INDICATOR IS ';TSI)
[5]   →0
[6]   ' '
[7]   ('SIMULATION RUN ENDED AT TIME ';CLOCK)
[8]   ('NUMBER TRANSACTIONS TERMINATED ';TRANS)
[9]   ' '
[10]  'ARE STATISTICS REQUIRED? YES OR NO'
[11]  →('N'=(INP□)[1])/0
[12]  STATISTICS
    ∇
```

```
      ∇NEWMODEL[□]∇

    ∇ NEWMODEL;R
[1]   →(0=R←COMP)/0
[2]   R←CONPAR
[3]   'TYPE NUMBER OF TRANSACTION PRIORITY CLASSES'
[4]   PRI←□
[5]   'TYPE NUMBER OF PARAMETERS FOR EACH TRANSACTION'
[6]   NOPAR←□
[7]   'TYPE NUMBER OF QUEUES REQUIRED IN MODEL'
[8]   NOQ←□
[9]   'TYPE NUMBER OF STORAGES REQUIRED FOLLOWED BY'
[10]  'CAPACITY OF EACH'
[11]  →(0=NOST←(,Z←□)[1])/13
[12]  STCAP←(⍳0),Z[1+⍳NOST]
[13]  'TYPE NUMBER OF FACILITIES REQUIRED'
[14]  NOF←□
[15]  NEWM30:INIT
[16]  'IF ADDITIONAL VARIABLES ARE REQUIRED IN THIS MODEL'
[17]  'THEY MUST BE SET UP BEFORE RUNNING THE MODEL'
    ∇
```

```
      ∇COMP[□]∇

    ∇ R←COMP
[1]   R←(I←⍳7)[1]
[2]   'TYPE NAMES OF ACTIVITIES TO BE USED IN THIS MODEL'
[3]   'EACH NAME CONTAINS 6 CHARACTERS SEPARATED BY A BLANK'
[4]   T←⍞,14⍴' ',□←' '
[5]   ' '
[6]   X←⍳0
[7]   L5:J←1
[8]   L10:→(0=∧/FCNS[J;]=T[I[⍳6]])/L20
[9]   X←X,J
[10]  →L5,I←I+7
[11]  L20:→((∧/T[I]='STOP   ')∨∧/T[I]='       ')/0
[12]  →((⍴FCNS)[1]≥J←J+1)/L10
[13]  ('ACTIVITY  ';T[I];' IS NOT IN MEMORY')
[14]  →R←0
    ∇


      ∇CONPAR[□]∇

    ∇ R←CONPAR;I
[1]   NEXT←((⍴X)[1],2)⍴0
[2]   'TYPE ONE CONFIGURATION PARAMETER FOR EACH ACTIVITY'
[3]   'TWO IF THE ACTIVITY CONTAINS AN ALTERNATE EXIT'
[4]   'A ''0'' INDICATES NO NEXT ACTIVITY'
[5]   R←I←1
[6]   L50:('ACTIVITY ';I)
[7]   NEXT[I;]←(□,0)[1 2]
[8]   →((⍴X)[1]≥I←I+1)/L50
    ∇


      ∇INIT[□]∇

    ∇ INIT;Z;R
[1]   CE←(1,NOPAR+6)⍴(NOPAR+6)⍺1
[2]   QUEUE←(NOQ,6)⍴0
[3]   STORAGE←(NOST,6)⍴0
[4]   FACIL←(NOF,6)⍴0
[5]   TRNO←TRANS←STOP←0
[6]   FECHAIN
[7]   'TYPE INITIAL VALUE OF SIMULATED TIME'
[8]   CLOCK←□
[9]   'TYPE LENGTH OF SIMULATION RUN'
[10]  ENDCLOCK←CLOCK+□
[11]  'TYPE MAXIMUM NUMBER OF TRANSACTIONS TO BE TERMINATED'
[12]  MAXTRANS←□
    ∇
```

```
      ∇FECHAIN[□]∇

    ∇ FECHAIN;N;NN
[1]   NN←6+NOPAR
[2]   'ARE ANY TRANSACTIONS TO BE PLACED ON FUTURE'
[3]   'EVENTS LIST? YES OR NO'
[4]   →('N'=⍞)/FECH20
[5]   'TYPE NUMBER OF TRANSACTIONS TO BE INPUT'
[6]   TRNO←N←□
[7]   ('TYPE ';NN;' VALUES FOR EACH TRANSACTION')
[8]   I←0
[9]   FE←(N,NN)ρ0
[10]  FECH5:→(NN≠ρR←□)/FECH10
[11]  FE[I←I+1;]←R
[12]  →((I<N),I≥N)/FECH5,FECH20
[13]  FECH10:→FECH5,ρ□←'RETYPE;INCORRECT NUMBER OF VALUES'
[14]  FECH20:'ARE VARIABLES MEAN AND MOD REQUIRED TO GENERA
      TE'
[15]  'NEW TRANSACTIONS? YES OR NO'
[16]  →(~GSW←~'N'=⍞[1])/0
[17]  'TYPE VALUES OF MEAN AND MOD'
[18]  MEAN←(R←□)[1]
[19]  MOD←R[2]
[20]  FE←(1,NOPAR+6)ρ1,(TRNO←TRNO+1),NEXT[1 2 ;1],
      1 4 ,NOPARρ0
    ∇


      ∇STATISTICS[□]∇

    ∇ STATISTICS;M;N
[1]   'ARE COLUMN TITLES REQUIRED? YES OR NO'
[2]   N←'N'=INP⍞[1]
[3]   '*FACILITY STATISTICS*'
[4]   →(NOF≠0)/STAT3
[5]   'NO FACILITIES IN THIS MODEL'
[6]   →STAT8
[7]   STAT3:→N/STAT5
[8]   'COL1 FACILITY NUMBER'
[9]   'COL2 AVERAGE UTILIZATION'
[10]  'COL3 TOTAL ENTRIES'
[11]  'COL4 AVERAGE TIME PER ENTRY'
[12]  STAT5:M←(NOF,4)ρ0
[13]  M[;1]←⍳NOF
[14]  M[;2]←FACIL[;5]÷FACIL[;6]
[15]  M[;3]←FACIL[;3]
[16]  M[;4]←FACIL[;5]÷FACIL[;3]
[17]  M
```

```
[18]  STAT8:'*STORAGE STATISTICS*',⎕←' '
[19]  →(NOST≠0)/STAT12
[20]  'NO STORAGES IN THIS MODEL'
[21]  →STAT14
[22]  STAT12:→N/STAT10
[23]  'COL1 STORAGE NUMBER'
[24]  'COL2 STORAGE CAPACITY'
[25]  'COL3 CURRENT CONTENTS'
[26]  'COL4 MAXIMUM CONTENTS'
[27]  'COL5 AVERAGE CONTENTS'
[28]  'COL6 AVERAGE UTILIZATION'
[29]  'COL7 TOTAL STORAGE UNITS ENTERED'
[30]  'COL8 AVERAGE TIME PER STORAGE UNIT'
[31]  STAT10:M←(NOST,8)⍴0
[32]  M[;1]←⍳NOST
[33]  M[;2]←STCAP
[34]  M[; 3 4]←STORAGE[; 1 2]
[35]  M[;5]←STORAGE[;5]÷STORAGE[;6]
[36]  M[;6]←M[;5]÷M[;2]
[37]  M[;7]←STORAGE[;3]
[38]  M[;8]←STORAGE[;5]÷STORAGE[;3]
[39]  M
[40]  STAT14:'*QUEUE STATISTICS*',⎕←' '
[41]  →(NOQ≠0)/STAT18
[42]  'NO QUEUES IN THIS MODEL'
[43]  →0
[44]  STAT18:→N/STAT15
[45]  'COL1 QUEUE NUMBER'
[46]  'COL2 CURRENT CONTENTS'
[47]  'COL3 MAXIMUM CONTENTS'
[48]  'COL4 AVERAGE CONTENTS'
[49]  'COL5 TOTAL ENTRIES'
[50]  'COL6 ZERO DELAY ENTRIES'
[51]  'COL7 AVERAGE TIME PER ENTRY'
[52]  'COL8 AVERAGE TIME PER ENTRY EXCLUDING'
[53]  '     ZERO DELAY ENTRIES'
[54]  STAT15:M←(NOQ,8)⍴0
[55]  M[;1]←⍳NOQ
[56]  M[; 2 3]←QUEUE[; 1 2]
[57]  M[;4]←QUEUE[;5]÷QUEUE[;6]
[58]  M[;5]←QUEUE[;3]
[59]  M[;6]←QUEUE[;4]
[60]  M[;7]←QUEUE[;5]÷QUEUE[;3]
[61]  M[;8]←QUEUE[;5]÷QUEUE[;3]-QUEUE[;4]
[62]  M
    ∇
```

```
      ∇LDST[□]∇

    ∇ R←LDST X;K;L;M;N;V;W;Y;Z
[1]   V←'VARIABLENAME          VARIABLENAME←'
[2]   K←'S'=X[1]
[3]   (X;' CURRENT EVENTS CHAIN BY TYPING')
[4]   □←(2 21 ρV,'CE',' ')[K+1;]
[5]   →K/LDST15
[6]   LDST10:CE←□
[7]   →(2=ρρCE)/LDST20
[8]   →LDST10,ρ□←N←'VARIABLENAME MUST BE A 2 DIMENSIONAL AR
      RAY'
[9]   LDST15:Z←□
[10]  LDST20:(X;' FUTURE EVENTS CHAIN BY TYPING')
[11]  □←(2 21 ρV,'FE',' ')[K+1;]
[12]  →K/LDST25
[13]  LDST23:FE←□
[14]  →(2=ρρFE)/LDST30
[15]  →LDST23,ρ□←N
[16]  LDST25:Z←□
[17]  LDST30:(X;' CONFIGURATION PARAMETERS BY TYPING')
[18]  □←(2 21 ρV,'NEXT',' ')[K+1;]
[19]  →K/LDST35
[20]  LDST33:NEXT←□
[21]  →(2=ρρNEXT)/LDST40
[22]  →LDST33,ρ□←N
[23]  LDST35:Z←□
[24]  LDST40:→K/LDST45
[25]  (X;' MODEL PARAMETERS BY TYPING')
[26]  □←(2 21 ρV,'X',' ')[K+1;]
[27]  LDST43:Z←□
[28]  CLOCK←Z[1]
[29]  ENDCLOCK←Z[2]
[30]  TRANS←Z[3]
[31]  MAXTRANS←Z[4]
[32]  MEAN←Z[5]
[33]  MOD←Z[6]
[34]  NOPAR←Z[7]
[35]  NOQ←Z[8]
[36]  NOST←Z[9]
[37]  NOF←Z[10]
[38]  TSI←Z[11]
[39]  PRI←Z[12]
[40]  STOP←Z[13]
[41]  GSW←Z[14]
[42]  TRNO←Z[15]
[43]  STCAP←Z[15+ι(W←15+NOST)-15]
[44]  X←Z[W+ι(ρZ)-W]
[45]  →LDST50
```

```
[46]  LDST45:Z←CLOCK,ENDCLOCK,TRANS,MAXTRANS,MEAN,MOD,NOPAR
[47]  X←Z,NOQ,NOST,NOF,TSI,PRI,STOP,GSW,TRNO,STCAP,X
[48]  'STORE MODEL PARAMETERS BY TYPING'
[49]  'VARIABLENAME←X'
[50]  Z←⍬
[51]  LDST50:(X;' QUEUE STATISTICS BY TYPING')
[52]  ⍞←(2 21 ⍴V,'QUEUE',' ')[K+1;]
[53]  →K/LDST55
[54]  QUEUE←⍬
[55]  →LDST60
[56]  LDST55:Z←⍬
[57]  LDST60:(X;' STORAGE STATISTICS BY TYPING')
[58]  ⍞←(2 21 ⍴V,'STORAGE',' ')[K+1;]
[59]  →K/LDST65
[60]  STORAGE←⍬
[61]  →LDST70
[62]  LDST65:Z←⍬
[63]  LDST70:(X;' FACILITY STATISTICS BY TYPING')
[64]  ⍞←(2 21 ⍴V,'FACIL',' ')[K+1;]
[65]  →K/LDST75
[66]  FACIL←⍬
[67]  →LDST80
[68]  LDST75:Z←⍬
[69]  LDST80:→K/LDST90
[70]  'IF ADDITIONAL VARIABLES ARE REQUIRED TO RUN THIS'
[71]  'MODEL,THEY SHOULD BE LOADED BY TYPING'
[72]  '  ''XXXX ← YYYY'''
[73]  'WHERE XXXX IS THE NAME OF THE VARIABLE USED'
[74]  'IN THE MODEL,AND YYYY IS THE NAME CURRENTLY'
[75]  'ASSOCIATED WITH THE VARIABLE'
[76]  ' '
[77]  'ARE ADDITIONAL VARIABLES REQUIRED? YES OR NO'
[78]  LDST85:→('N'=⍞[1])/0
[79]  'TYPE INSTRUCTION'
[80]  Z←⍬
[81]  →LDST85,⍴⍞←'MORE? YES OR NO'
[82]  LDST90:'IF ADDITIONAL VARIABLES HAVE BEEN USED'
[83]  'IN THIS MODEL, THEY SHOULD BE STORED BY TYPING'
[84]  '  ''XXXX ← YYYY'''
[85]  'WHERE VARIABLE YYYY IS TO BE STORED IN'
[86]  'VARIABLE XXXX'
[87]  'ANY ADDITIONAL VARIABLES? YES OR NO'
[88]  LDST95:→('N'=⍞[1])/0
[89]  'TYPE INSTRUCTION'
[90]  Z←⍬
[91]  →LDST95,⍴⍞←'MORE? YES OR NO'
    ∇
```

SCHEDULING PROGRAMS

```
      ∇SCAN[□]∇

    ∇ CHK←SCAN;IND;IND1;I;K;KNT;L
[1]   CHK←1
[2]   SCAN25:SCF←0
[3]   →(((+/CE[;6]=1)=ρIND)∨0=ρIND←((4=CE[;6])∨1=CE[;
      6])/ι(ρCE)[1])/SCAN100
[4]   KNT←1
[5]   SCAN30:CE[CURTR←IND[KNT];6]←TSI←4
[6]   SCAN35:L←BRANCH X[CE[CURTR;3]]
[7]   →(1=STOP)/0
[8]   →(TSI=ι5)/SCAN90,SCAN120,SCAN120,SCAN80,SCAN120
[9]   →CHK←0
[10]  SCAN80:K←NEXTFAC CURTR
[11]  →SCAN35
[12]  SCAN90:→((SCF=1),(~K),K←(ρIND)<KNT←KNT+1)/SCAN25,
      SCAN30,SCAN25
[13]  SCAN100:FETOCE IND←(FE[;5]=⌊/FE[;5])/ι(ρFE)[1]
[14]  →(1=STOP)/0
[15]  →SCAN25
[16]  SCAN120:IND←((4=CE[;6])∨1=CE[;6])/ι(ρCE)[1]
[17]  →SCAN90
    ∇
```

```
      ∇FETOCE[□]∇

    ∇ FETOCE V;M;N;P;X
[1]   X←V
[2]   →(ENDCLOCK≥⌊/FE[;5])/FETOCE1
[3]   →~STOP←1
[4]   FETOCE1:N←(((ι(ρFE)[1])∈V)∧FE[;1]∈M←⌊/FE[V;1])/ι(ρFE)
      [1]
[5]   P←+/[1]CE[;1]∘.=ιPRI
[6]   CE←(((+/P[ιM])ρ1),((ρN)ρ0),((+/P)-+/P[ιM])ρ1)\[1]CE
[7]   FE[N;6]←4
[8]   CE[(ιρN)++/P[ιM];]←FE[N;]
[9]   →(0≠ρV←(V≠N)/V)/FETOCE1
[10]  CLOCK←⌊/FE[;5]
[11]  →((0=∨/NEXT[1;1]=FE[X;3])∨GSW=0)/FETOCE2
[12]  V←MEAN GENERATE MOD
[13]  FETOCE2:V←(ρFE)[1]ρ1
[14]  V[X]←0
[15]  FE←V/[1]FE
    ∇
```

```
      ∇MOVECTOF[□]∇

    ∇ R←MOVECTOF K;M;V
[1]   FE←(M←((ρFE)[1]ρ1),0)\[1]FE
[2]   FE[ρM;]←CE[K;]
[3]   V←((ρCE)[1])ρ1
[4]   R←V[K]←0
[5]   CE←V/[1]CE
    ∇
```

```
      ∇NEXTFAC[□]∇

    ∇ R←NEXTFAC I;K;L
[1]   R←K←CE[I;3]←CE[I;4]
[2]   →(0≠NEXT[K;2])/0
[3]   CE[I;4]←NEXT[K;1]
    ∇
```

```
      ∇GENERATE[□]∇

    ∇ R←M GENERATE MOD;I
[1]   FE←(Iα(I←ρ((ρFE)[1]ρ1),0)-1)\[1]FE
[2]   FE[I;]←1,(TRNO←TRNO+1),NEXT[1;1],NEXT[2;1],(CLOCK+M),
      4,NOPARρ0
[3]   →(R←0=MOD)/0
[4]   FE[I;5]←CLOCK+M RAND MOD
    ∇
```

TRANSACTION STATUS PROGRAMS

```
      ∇ADVANCE[□]∇

    ∇ ADVANCE A;R
[1]   CE[CURTR;5]←CLOCK+⌊A
[2]   TSI←3
[3]   R←NEXTFAC CURTR
[4]   →(0=⌊A)/0
[5]   →MOVECTOF CURTR
    ∇
```

```
      ∇BLOCK[□]∇

    ∇ BLOCK
[1]   CE[CURTR;6]←TSI←1
    ∇
```

```
      ∇HOLD[□]∇

    ∇ HOLD A
[1]   CE[CURTR;5]←CLOCK+⌊A
[2]   TSI←2
[3]   →MOVECTOF CURTR
    ∇
```

```
      ∇ENDTRANS[□]∇

    ∇ ENDTRANS;V
[1]   TRANS←TRANS+1
[2]   V←(ρCE)[1]ρ1
[3]   V[CURTR]←0
[4]   CE←V/[1]CE
[5]   →((TRANS≥MAXTRANS)∨(CLOCK≥ENDCLOCK))/8
[6]   TSI←5
[7]   →0
[8]   TSI←5+STOP←1
    ∇
```

UTILITY PROGRAMS

```
      ∇FACILITYIN[□]∇

    ∇ R←FACILITYIN A
[1]   R←0
[2]   →(FACIL[A;1]=0)/5
[3]   BLOCK
[4]   →~R←1
[5]   FACIL←FACIL STATIN A,1
    ∇
```

```
      ∇FACILITYOUT[□]∇

    ∇ FACILITYOUT A
[1]   FACIL←FACIL STATOUT A,1
[2]   SCF←1
    ∇
```

```
      ∇STORAGEIN[□]∇

    ∇ R←A STORAGEIN B
[1]   R←0
[2]   →(STCAP[A]≥STORAGE[A;1]+B)/ST1
[3]   BLOCK
[4]   →~R←1
[5]   ST1:STORAGE←STORAGE STATIN A,B
    ∇
```

```
      ∇STORAGEOUT[□]∇

    ∇ A STORAGEOUT B
[1]   STORAGE←STORAGE STATOUT A,B
[2]   SCF←1
    ∇
```

```
      ∇QUEUEIN[□]∇

    ∇ A QUEUEIN B
[1]   QUEUE←QUEUE STATIN A,B
    ∇


      ∇QUEUEOUT[□]∇

    ∇ A QUEUEOUT B
[1]   QUEUE←QUEUE STATOUT A,B
    ∇


      ∇ASGN[□]∇

    ∇ A ASGN B
[1]   CE[CURTR;A+6]←B
    ∇


      ∇TRANSFER[□]∇

    ∇ TRANSFER A;B
[1]   B←CE[CURTR;3]
[2]   →A/5
[3]   CE[CURTR;4]←NEXT[B;2]
[4]   →0
[5]   CE[CURTR;4]←NEXT[B;1]
    ∇


      ∇PRIORITY[□]∇

    ∇ PRIORITY A
[1]   CE[CURTR;1]←A
    ∇


      ∇RAND[□]∇

    ∇ R←M RAND N
[1]   →(0=N)/RAND1
[2]   →0,R←((M-N)-1)+?1+N×2
[3]   RAND1:R←M
    ∇
```

```
      ∇QCONT[□]∇

    ∇ R←QCONT A
[1]   R←QUEUE[A;1]
    ∇


      ∇STCONT[□]∇

    ∇ R←STCONT A
[1]   R←STORAGE[A;1]
    ∇


      ∇FACILITY[□]∇

    ∇ R←FACILITY A
[1]   R←FACIL[A;1]
    ∇


      ∇STATIN[□]∇

    ∇ R←M STATIN N
[1]   M[N[1];5]←M[N[1];5]+M[N[1];1]×CLOCK-M[N[1];6]
[2]   M[N[1];1]←M[N[1];1]+N[2]
[3]   M[N[1];2]←⌈/M[N[1];1],M[N[1];2]
[4]   M[N[1];3]←M[N[1];3]+N[2]
[5]   M[N[1];6]←CLOCK
[6]   R←M
    ∇


      ∇STATOUT[□]∇

    ∇ R←M STATOUT N
[1]   M[N[1];4]←M[N[1];4]+CLOCK=M[N[1];6]
[2]   M[N[1];5]←M[N[1];5]+M[N[1];1]×CLOCK-M[N[1];6]
[3]   M[N[1];1]←M[N[1];1]-N[2]
[4]   M[N[1];6]←CLOCK
[5]   R←M
    ∇


      ∇PARAMETER[□]∇

    ∇ R←PARAMETER A
[1]   R←CE[CURTR;6+A]
    ∇
```

EXAMPLE - SECTION 4.3

```
      ∇MSGIN[□]∇

    ∇ MSGIN
[1]   1 ASGN(?10)
[2]   2 ASGN(?10)
[3]   →((PARAMETER 1)=PARAMETER 2)/2
[4]   3 ASGN 6 RAND 2
[5]   1 QUEUEIN 1
    ∇
```

```
      ∇TELCAL[□]∇

    ∇ TELCAL
[1]   →((FACILITY PARAMETER 1)∨FACILITY PARAMETER
      2)/10
[2]   →(0=ρLINK←(~FACILITY 10+ι5)/10+ι5)/10
[3]   1 QUEUEOUT 1
[4]   R←FACILITYIN PARAMETER 1
[5]   R←FACILITYIN PARAMETER 2
[6]   R←FACILITYIN LINK[1]
[7]   4 ASGN LINK[1]
[8]   ADVANCE PARAMETER 3
[9]   →0
[10]  BLOCK
    ∇
```

```
      ∇ENDCAL[□]∇

    ∇ ENDCAL
[1]   FACILITYOUT PARAMETER 1
[2]   FACILITYOUT PARAMETER 2
[3]   FACILITYOUT PARAMETER 4
[4]   ENDTRANS
    ∇
```

```
      OSS

ENTER COMMAND WORD
NEWMODEL

TYPE NAMES OF ACTIVITIES TO BE USED IN THIS MODEL
EACH NAME CONTAINS 6 CHARACTERS SEPARATED BY A BLANK

GENRAT MSGIN  TELCAL ENDCAL

TYPE ONE CONFIGURATION PARAMETER FOR EACH ACTIVITY
TWO IF THE ACTIVITY CONTAINS AN ALTERNATE EXIT
A '0' INDICATES NO NEXT ACTIVITY
ACTIVITY 1
□:
      2
ACTIVITY 2
□:
      3
ACTIVITY 3
□:
      4
ACTIVITY 4
□:
      0
TYPE NUMBER OF TRANSACTION PRIORITY CLASSES
□:
      1
TYPE NUMBER OF PARAMETERS FOR EACH TRANSACTION
□:
      4
TYPE NUMBER OF QUEUES REQUIRED IN MODEL
□:
      1
TYPE NUMBER OF STORAGES REQUIRED FOLLOWED BY
CAPACITY OF EACH
□:
      0
TYPE NUMBER OF FACILITIES REQUIRED
□:
      15
ARE ANY TRANSACTIONS TO BE PLACED ON FUTURE
EVENTS LIST? YES OR NO
NO
ARE VARIABLES MEAN AND MOD REQUIRED TO GENERATE
NEW TRANSACTIONS? YES OR NO
YES
```

```
TYPE VALUES OF MEAN AND MOD
□:
       6    2
TYPE INITIAL VALUE OF SIMULATED TIME
□:
       0
TYPE LENGTH OF SIMULATION RUN
□:
       200
TYPE MAXIMUM NUMBER OF TRANSACTIONS TO BE TERMINATED
□:
       50
IF ADDITIONAL VARIABLES ARE REQUIRED IN THIS MODEL
THEY MUST BE SET UP BEFORE RUNNING THE MODEL

ENTER COMMAND WORD
RUN


SIMULATION RUN ENDED AT TIME 199
NUMBER TRANSACTIONS TERMINATED 30

ARE STATISTICS REQUIRED? YES OR NO
YES
ARE COLUMN TITLES REQUIRED? YES OR NO
YES
```

```
*FACILITY STATISTICS*
COL1 FACILITY NUMBER
COL2 AVERAGE UTILIZATION
COL3 TOTAL ENTRIES
COL4 AVERAGE TIME PER ENTRY

   1          0.3403     10          6.5
   2          0.2563      8          6.375
   3          0.3194     10          6.1
   4          0.1538      4          5.5
   5          0.1189      4          5.5
   6          0.1923      6          5.833
   7          0.06349     1          8
   8          0.3065     10          6.1
   9          0.1436      5          5.2
  10          0.1507      2          5.5
  11          0.5879     19          6.158
  12          0.3315     10          6
  13          0.02162     1          4
  14          1           0          1
  15          1           0          1

*STORAGE STATISTICS*
NO STORAGES IN THIS MODEL

*QUEUE STATISTICS*
COL1 QUEUE NUMBER
COL2 CURRENT CONTENTS
COL3 MAXIMUM CONTENTS
COL4 AVERAGE CONTENTS
COL5 TOTAL ENTRIES
COL6 ZERO DELAY ENTRIES
COL7 AVERAGE TIME PER ENTRY
COL8 AVERAGE TIME PER ENTRY EXCLUDING
     ZERO DELAY ENTRIES

   1          0          1          0.0829    30
        27          0.5333     5.333

ENTER COMMAND WORD
END

SIMULATION COMPLETED
```

APPENDIX III

RESULTS OF OSS APPLICATIONS

The following pages contain listings of the results obtained when the systems described in Chapter V were simulated using OSS. For each of the three systems described the results consist of:

1. listings of the activities used in the model,
2. listings describing the assembling and running of the model, and
3. listings of the statistical results obtained in that run.

THREE MACHINE ASSEMBLY SYSTEM

```
      ∇REJECT[□]∇

    ∇ REJECT
[1]   1 ASGN 2
[2]   →(9≥?10)/4
[3]   1 ASGN(PARAMETER 1)-1
[4]   →(9≥?10)/7
[5]   1 ASGN(PARAMETER 1)-1
[6]   →(0=PARAMETER 1)/12
[7]   →(2=PARAMETER 1)/9
[8]   →((STCONT 1)÷2)=⌈(STCONT 1)÷2)/11
[9]   R←1 STORAGEIN PARAMETER 1
[10]  →0
[11]  R←1 STORAGEIN 1
[12]  ENDTRANS
    ∇


      ∇MACHNC[□]∇

    ∇ MACHNC
[1]   →(FACILITYIN 1)/0
[2]   1 STORAGEOUT 2
[3]   ADVANCE 15
    ∇


      ∇ENDTRN[□]∇

    ∇ ENDTRN
[1]   R←2 STORAGEIN 1
[2]   FACILITYOUT 1
[3]   ENDTRANS
    ∇
```

```
      OSS

ENTER COMMAND WORD
NEWMODEL

TYPE NAMES OF ACTIVITIES TO BE USED IN THIS MODEL
EACH NAME CONTAINS 6 CHARACTERS SEPARATED BY A BLANK

GENRAT REJECT MACHNC ENDTRN

TYPE ONE CONFIGURATION PARAMETER FOR EACH ACTIVITY
TWO IF THE ACTIVITY CONTAINS AN ALTERNATE EXIT
A '0' INDICATES NO NEXT ACTIVITY
ACTIVITY 1
□:
      2
ACTIVITY 2
□:
      3
ACTIVITY 3
□:
      4
ACTIVITY 4
□:
      0
TYPE NUMBER OF TRANSACTION PRIORITY CLASSES
□:
      1
TYPE NUMBER OF PARAMETERS FOR EACH TRANSACTION
□:
      1
TYPE NUMBER OF QUEUES REQUIRED IN MODEL
□:
      0
TYPE NUMBER OF STORAGES REQUIRED FOLLOWED BY
CAPACITY OF EACH
□:
      2   500   250
TYPE NUMBER OF FACILITIES REQUIRED
□:
      1
ARE ANY TRANSACTIONS TO BE PLACED ON FUTURE
EVENTS LIST? YES OR NO
NO
ARE VARIABLES MEAN AND MOD REQUIRED TO GENERATE
NEW TRANSACTIONS? YES OR NO
YES
```

```
TYPE VALUES OF MEAN AND MOD
□:
      20    0
TYPE INITIAL VALUE OF SIMULATED TIME
□:
      0
TYPE LENGTH OF SIMULATION RUN
□:
      480
TYPE MAXIMUM NUMBER OF TRANSACTIONS TO BE TERMINATED
□:
      1000
IF ADDITIONAL VARIABLES ARE REQUIRED IN THIS MODEL
THEY MUST BE SET UP BEFORE RUNNING THE MODEL

ENTER COMMAND WORD
RUN


SIMULATION RUN ENDED AT TIME 476
NUMBER TRANSACTIONS TERMINATED 24

ARE STATISTICS REQUIRED? YES OR NO
YES
ARE COLUMN TITLES REQUIRED? YES OR NO
YES
```

```
*FACILITY STATISTICS*
COL1 FACILITY NUMBER
COL2 AVERAGE UTILIZATION
COL3 TOTAL ENTRIES
COL4 AVERAGE TIME PER ENTRY

    1          0.5987    19          15

*STORAGE STATISTICS*
COL1 STORAGE NUMBER
COL2 STORAGE CAPACITY
COL3 CURRENT CONTENTS
COL4 MAXIMUM CONTENTS
COL5 AVERAGE CONTENTS
COL6 AVERAGE UTILIZATION
COL7 TOTAL STORAGE UNITS ENTERED
COL8 AVERAGE TIME PER STORAGE UNIT

  1.000E0     5.000E2     0.000E0     3.000E0     5.206E-1
        1.041E-3    3.800E1     6.316E0
  2.000E0     2.500E2     1.900E1     1.900E1     8.319E0
        3.328E-2    1.900E1     2.084E2

*QUEUE STATISTICS*
NO QUEUES IN THIS MODEL

ENTER COMMAND WORD
END

SIMULATION COMPLETED
```

REAL-TIME DATA PROCESSING SYSTEM

```
      ∇MSGIN[□]∇

    ∇ MSGIN
[1]   →(FACILITYIN 5)/0
[2]   1 ASGN 15 RAND 10
[3]   2 ASGN(?3)
[4]   ADVANCE 1
    ∇
```

```
      ∇FACOUT[□]∇

    ∇ FACOUT
[1]   FACILITYOUT 5
    ∇
```

```
      ∇STORIN[□]∇

    ∇ STORIN
[1]   →(1 STORAGEIN PARAMETER 1)/0
[2]   (PARAMETER 2)QUEUEIN 1
    ∇
```

```
      ∇QTEST[□]∇

    ∇ QTEST
[1]   →(FACILITYIN PARAMETER 2)/0
[2]   (PARAMETER 2)QUEUEOUT 1
[3]   ADVANCE 120 RAND 80
    ∇
```

```
      ∇ADVANC[□]∇

    ∇ ADVANC
[1]   ADVANCE 25 RAND 25
    ∇
```

```
      ∇READIS[□]∇

    ∇ READIS
[1]   →(FACILITYIN 4)/4
[2]   ADVANCE 5
[3]   →0
[4]   HOLD 50
    ∇
```

```
      ∇ENDISK[□]∇

    ∇ ENDISK
[1]   FACILITYOUT 4
[2]   FACILITYOUT PARAMETER 2
    ∇
```

```
      ∇PROMES[□]∇

    ∇ PROMES
[1]   →(FACILITYIN 5)/0
[2]   ADVANCE 0.4×PARAMETER 1
    ∇
```

```
      ∇ENDMES[□]∇

    ∇ ENDMES
[1]   FACILITYOUT 5
[2]   1 STORAGEOUT PARAMETER 1
[3]   ENDTRANS
    ∇
```

```
      OSS

ENTER COMMAND WORD
NEWMODEL

TYPE NAMES OF ACTIVITIES TO BE USED IN THIS MODEL
EACH NAME CONTAINS 6 CHARACTERS SEPARATED BY A BLANK

GENRAT MSGIN  FACOUT STORIN QTEST  ADVANC READIS
ENDISK PROMES ENDMES

TYPE ONE CONFIGURATION PARAMETER FOR EACH ACTIVITY
TWO IF THE ACTIVITY CONTAINS AN ALTERNATE EXIT
A '0' INDICATES NO NEXT ACTIVITY
ACTIVITY 1
□:
      2
ACTIVITY 2
□:
      3
ACTIVITY 3
□:
      4
ACTIVITY 4
□:
      5
ACTIVITY 5
□:
      6
ACTIVITY 6
□:
      7
ACTIVITY 7
□:
      8
ACTIVITY 8
□:
      9
ACTIVITY 9
□:
      10
ACTIVITY 10
□:
      0
TYPE NUMBER OF TRANSACTION PRIORITY CLASSES
□:
      1
TYPE NUMBER OF PARAMETERS FOR EACH TRANSACTION
□:
      2
```

```
TYPE NUMBER OF QUEUES REQUIRED IN MODEL
□:
      3
TYPE NUMBER OF STORAGES REQUIRED FOLLOWED BY
CAPACITY OF EACH
□:
      1   1500
TYPE NUMBER OF FACILITIES REQUIRED
□:
      5
ARE ANY TRANSACTIONS TO BE PLACED ON FUTURE
EVENTS LIST? YES OR NO
NO
ARE VARIABLES MEAN AND MOD REQUIRED TO GENERATE
NEW TRANSACTIONS? YES OR NO
YES
TYPE VALUES OF MEAN AND MOD
□:
      50   25
TYPE INITIAL VALUE OF SIMULATED TIME
□:
      0
TYPE LENGTH OF SIMULATION RUN
□:
      3000
TYPE MAXIMUM NUMBER OF TRANSACTIONS TO BE TERMINATED
□:
      100
IF ADDITIONAL VARIABLES ARE REQUIRED IN THIS MODEL
THEY MUST BE SET UP BEFORE RUNNING THE MODEL

ENTER COMMAND WORD
RUN


SIMULATION RUN ENDED AT TIME 2995
NUMBER TRANSACTIONS TERMINATED 47

ARE STATISTICS REQUIRED? YES OR NO
YES
ARE COLUMN TITLES REQUIRED? YES OR NO
YES
```

```
*FACILITY STATISTICS*
COL1 FACILITY NUMBER
COL2 AVERAGE UTILIZATION
COL3 TOTAL ENTRIES
COL4 AVERAGE TIME PER ENTRY

   1          0.9461     19          143.3
   2          0.784      16          141.1
   3          0.7571     15          143.2
   4          0.0818     47            5
   5          0.1112    109            3.055

*STORAGE STATISTICS*
COL1 STORAGE NUMBER
COL2 STORAGE CAPACITY
COL3 CURRENT CONTENTS
COL4 MAXIMUM CONTENTS
COL5 AVERAGE CONTENTS
COL6 AVERAGE UTILIZATION
COL7 TOTAL STORAGE UNITS ENTERED
COL8 AVERAGE TIME PER STORAGE UNIT

  1.000E0     1.500E3     1.650E2     2.020E2     1.309E2
        8.727E-2    8.920E2     4.395E2

*QUEUE STATISTICS*
COL1 QUEUE NUMBER
COL2 CURRENT CONTENTS
COL3 MAXIMUM CONTENTS
COL4 AVERAGE CONTENTS
COL5 TOTAL ENTRIES
COL6 ZERO DELAY ENTRIES
COL7 AVERAGE TIME PER ENTRY
COL8 AVERAGE TIME PER ENTRY EXCLUDING
     ZERO DELAY ENTRIES

    1          10          10          4.292      29
           1         443.3       459.1
    2           1           3          0.5891     17
           6         101.5       156.9
    3           1           4          1.467      16
           3         271.3       333.9

ENTER COMMAND WORD
END

SIMULATION COMPLETED
```

BRANCH LIBRARY SYSTEM

```
      ∇ENTER[□]∇

    ∇ ENTER
[1]   →(8≤QCONT 1)/4
[2]   1 QUEUEIN 1
[3]   →0
[4]   ENDTRANS
    ∇
```

```
      ∇INFORM[□]∇

    ∇ INFORM
[1]   →(FACILITYIN 1)/0
[2]   1 QUEUEOUT 1
[3]   ADVANCE 8 RAND 3
    ∇
```

```
      ∇SPLIT[□]∇

    ∇ SPLIT
[1]   FACILITYOUT 1
[2]   1 ASGN(?10)
[3]   TRANSFER 4>PARAMETER 1
[4]   →(4>PARAMETER 1)/11
[5]   →((QCONT 3)>QCONT 4)/8
[6]   1 ASGN 3
[7]   →9
[8]   1 ASGN 4
[9]   (PARAMETER 1)QUEUEIN 1
[10]  →0
[11]  →(2<QCONT 2)/14
[12]  2 QUEUEIN 1
[13]  →0
[14]  ENDTRANS
    ∇
```

```
      ∇BOOKLI[□]∇

    ∇ BOOKLI
[1]   →(FACILITYIN 2)/0
[2]   2 QUEUEOUT 1
[3]   ADVANCE 25 RAND 5
    ∇
```

```
      ∇DUPMAC[□]∇

    ∇ DUPMAC
[1]   FACILITYOUT 2
[2]   →((6<?10)∨1=FACILITY 3)/6
[3]   R←FACILITYIN 3
[4]   ADVANCE 45 RAND 10
[5]   →0
[6]   ENDTRANS
    ∇
```

```
      ∇ENDTR2[□]∇

    ∇ ENDTR2
[1]   FACILITYOUT 3
[2]   ENDTRANS
    ∇
```

```
      ∇DOCLIB[□]∇

    ∇ DOCLIB
[1]   →(FACILITYIN 1+PARAMETER 1)/0
[2]   (PARAMETER 1)QUEUEOUT 1
[3]   →(4=PARAMETER 1)/6
[4]   ADVANCE 18 RAND 3
[5]   →0
[6]   ADVANCE 22 RAND 4
    ∇
```

```
      ∇FILMQU[□]∇

    ∇ FILMQU
[1]   FACILITYOUT 1+PARAMETER 1
[2]   →((25<?100)∨3≤STCONT 1)/6
[3]   R←1 STORAGEIN 1
[4]   5 QUEUEIN 1
[5]   →0
[6]   ENDTRANS
    ∇
```

```
      ∇READER[□]∇

    ∇ READER
[1]   →(FACILITYIN 6)/0
[2]   5 QUEUEOUT 1
[3]   ADVANCE 35 RAND 5
    ∇
```

```
      ∇ENDTR1[□]∇

    ∇ ENDTR1
[1]   FACILITYOUT 6
[2]   1 STORAGEOUT 1
[3]   ENDTRANS
    ∇
```

```
      OSS

ENTER COMMAND WORD
NEWMODEL

TYPE NAMES OF ACTIVITIES TO BE USED IN THIS MODEL
EACH NAME CONTAINS 6 CHARACTERS SEPARATED BY A BLANK

GENRAT ENTER  INFORM SPLIT  BOOKLI DUPMAC ENDTR2
DOCLIB FILMQU READER ENDTR1

TYPE ONE CONFIGURATION PARAMETER FOR EACH ACTIVITY
TWO IF THE ACTIVITY CONTAINS AN ALTERNATE EXIT
A '0' INDICATES NO NEXT ACTIVITY
ACTIVITY 1
□:
      2
ACTIVITY 2
□:
      3
ACTIVITY 3
□:
      4
ACTIVITY 4
□:
      5   8
ACTIVITY 5
□:
      6
ACTIVITY 6
□:
      7
ACTIVITY 7
□:
      0
ACTIVITY 8
□:
      9
ACTIVITY 9
□:
      10
ACTIVITY 10
□:
      11
ACTIVITY 11
□:
      0
TYPE NUMBER OF TRANSACTION PRIORITY CLASSES
□:
      1
```

```
TYPE NUMBER OF PARAMETERS FOR EACH TRANSACTION
□:
      1
TYPE NUMBER OF QUEUES REQUIRED IN MODEL
□:
      5
TYPE NUMBER OF STORAGES REQUIRED FOLLOWED BY
CAPACITY OF EACH
□:
      1   3
TYPE NUMBER OF FACILITIES REQUIRED
□:
      6
ARE ANY TRANSACTIONS TO BE PLACED ON FUTURE
EVENTS LIST? YES OR NO
NO
ARE VARIABLES MEAN AND MOD REQUIRED TO GENERATE
NEW TRANSACTIONS? YES OR NO
YES
TYPE VALUES OF MEAN AND MOD
□:
      5   2
TYPE INITIAL VALUE OF SIMULATED TIME
□:
      0
TYPE LENGTH OF SIMULATION RUN
□:
      1000
TYPE MAXIMUM NUMBER OF TRANSACTIONS TO BE TERMINATED
□:
      100
IF ADDITIONAL VARIABLES ARE REQUIRED IN THIS MODEL
THEY MUST BE SET UP BEFORE RUNNING THE MODEL

ENTER COMMAND WORD
RUN


SIMULATION RUN ENDED AT TIME 570
NUMBER TRANSACTIONS TERMINATED 100

ARE STATISTICS REQUIRED? YES OR NO
YES
ARE COLUMN TITLES REQUIRED? YES OR NO
YES
```

```
*FACILITY STATISTICS*
COL1 FACILITY NUMBER
COL2 AVERAGE UTILIZATION
COL3 TOTAL ENTRIES
COL4 AVERAGE TIME PER ENTRY

     1          0.7743     56          7.839
     2          0.7528     16         25.13
     3          0.5189      5         41.2
     4          0.8268     26         17.81
     5          0.4138     11         20.73
     6          0.3285      6         30.17

*STORAGE STATISTICS*
COL1 STORAGE NUMBER
COL2 STORAGE CAPACITY
COL3 CURRENT CONTENTS
COL4 MAXIMUM CONTENTS
COL5 AVERAGE CONTENTS
COL6 AVERAGE UTILIZATION
COL7 TOTAL STORAGE UNITS ENTERED
COL8 AVERAGE TIME PER STORAGE UNIT

     1          3          2          3          0.4229
           0.141      7          33.29

*QUEUE STATISTICS*
COL1 QUEUE NUMBER
COL2 CURRENT CONTENTS
COL3 MAXIMUM CONTENTS
COL4 AVERAGE CONTENTS
COL5 TOTAL ENTRIES
COL6 ZERO DELAY ENTRIES
COL7 AVERAGE TIME PER ENTRY
COL8 AVERAGE TIME PER ENTRY EXCLUDING
     ZERO DELAY ENTRIES

   1          8          8          7.534      64
         1          66.75      67.81
   2          0          3          1.352      16
         3          42.75      52.62
   3          1          1          0.4442     27
         7           9.296     12.55
   4          0          1          0.1125     11
         6           5.636     12.4
   5          1          2          0.09437     7
         4           7.429     17.33

ENTER COMMAND WORD
END

SIMULATION COMPLETED
```